7/31/02

Examiner Siek,

AU 2895
CP 3 E08

Re: 09/246,047

Please find attached edited first-pass search results from the patent and non-patent commercial abstract and full-text databases. The search strategy was based on the claims and the statements of the technical problems and solutions. Tagged records might be worth your review as well as the rest of the references provided.

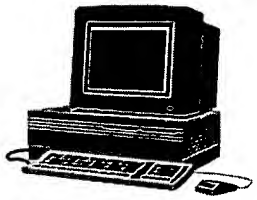
If you have any further questions, please let me know.

Thank You,

Irina Speckhard
308-6559
STIC-EIC2800
CP4-9C18

EIC search Report

EIC2800
Search Results
Feedback Form (Optional)



Scientific & Technical Information Center

The search results generated for your recent request are attached. If you have any questions or comments (compliments or complaints) about the scope or the results of the search, please contact ***the EIC searcher*** who conducted the search ***or contact***:

Jeff Harrison, Team Leader, 306-5429

Voluntary Results Feedback Form

- *I am an examiner in Workgroup:* [ ] *Example:* 2830

- *Relevant prior art **found**, search results used as follows:*
  - ☐ 102 rejection
  - ☐ 103 rejection
  - ☐ Cited as being of interest.
  - ☐ Helped examiner better understand the invention.
  - ☐ Helped examiner better understand the state of the art in their technology.

Types of relevant prior art found:

- ☐ Foreign Patent(s)
- ☐ Non-Patent Literature
  (journal articles, conference proceedings, new product announcements etc.)

- *Relevant prior art **not found**:*
  - ☐ Results verified the lack of relevant prior art (helped determine patentability).
  - ☐ Search results were not useful in determining patentability or understanding the invention.

Other Comments:

Drop off completed forms in **CP4-9C18,** or send to **Jeff Harrison, CP4-9C18.**

Abstract

nML is a formalism targetted for describing arbitrary single-processor computer architectures. nML works at the instruction set level, i.e. it hides implementation issues of the actual machine. nML can be used as an input language for a wide range of tools that need formal machine descriptions. Based on attribute grammars, nML is flexible and reasonably easy to use.

Contents

1 Introduction

1.1 Where are Machine Descriptions needed?

There are a lot of different software applications where detailed formal descriptions of computer architectures are needed. A few of these are:

- Simulators:
  in the development phase of an architecture, i.e. before actual hardware exists, instruction-level simulations are needed for writing the first actual programs and for testing compiler code generation.

- Assemblers and Dissassemblers:
  these programs are so simple that they ought to be easily generated automatically once a formal description of the assembler syntax and binary coding of the instruction set exist.

- Compiler back ends:
  modern compiler technology uses pattern-matched code-generation schemes. These patterns are usually written and optimized by hand and serve as "indirect" machine descriptions; the rest of the knowledge about the machine (e.g. pipeline behaviour and register allocation schemes) is hand-coded into the pattern-matcher or into special allocation and optimization passes. It should be possible to generate code-generation pattern libraries out of "functional" machine descriptions, i.e. one that are not centered on the special needs of the code generator. A more ambitious goal would be the generation of the whole code generator.

1.2 Different Kinds of Machine Descriptions

Many different kinds of machine descriptions are employed today. Two of the better known ones include:

1.2.1 GCC's .md format

GCC, the GNU C compiler, can be adapted to different machines by changing its machine description[1]. To quote from [9]:

> GNU CC gets most of the information about the target machine from a machine description which gives an algebraic formula for each of the machine's instructions. This is a very clean way to describe the target. But when the compiler needs information that is difficult to express in this fashion, I have not hesitated to define an ad-hoc parameter to the machine description.
>
> [...]
>
> A machine description has two parts: a file of instruction patterns ('.md' file) and a C header file of macro definitions.
>
> The '.md' file for a target machine contains a pattern for each instruction that the target machine supports (or at least each instruction that is worth telling the compiler about).
>
> [...]
>
> Each instruction pattern contains an incomplete RTL expression, with pieces to be filled in later, operand constraints that restrict how the pieces can be filled in, and an output pattern or C code to generate the assembler output, all wrapped up in a '`define_insn`' expression.
>
> [...]
>
> Here is an actual example of an instruction pattern, for the 68000/68020.

```
(define_insn "tstsi"
  [(set (cc0)
        (match_operand:SI 0 "general_operand" "rm"))]
  ""
  "*
 {if (TARGET_68020 || ! ADDRESS_REG_P (operands[0]))
    return \"tstl %0\";
  return \"cmpl #0,%0\"; }")
```

[1] At least when the machines don't derive too much from the built-in assumptions

RTL is the intermediate language of the compiler. A special program transforms a machine description into a C function that is used within the compiler. As can be seen in the example, the description, while being powerful – after all, any C function may be incorporated into the pattern-matching and expanding process –, is quite dependent on compiler internals and not very intuitive.

1.2.2 The VHDL Hardware Definition Language

VHDL is a language used for describing all kinds of digital circuits, among them processors and their components. A VHDL 'program' describes a circuit as a box having a number of input and output ports either by assembling it from other, previously defined boxes, or by giving 'the program that executes inside' the box. When using VHDL to describe a computer architecture, usually the whole data-path-, fetch-, decode-, load-, execute- and store-mechanism is described. In the simplest case, the processor is seen as a black box containing a large 'switch'-statement, i.e. the description really is a simulation program.

There are a number of other languages similar to VHDL, most notably the "Electronic Design Interchange Format" EDIF, which mainly describes the graphical layout of a circuit and has no 'semantic level' aside from the information about connections between predefined cells, and HILARICS-2, which is mostly equivalent to a human-readable version of the 'net'-part of EDIF.

VHDL is described in [2], EDIF in [5, 6] and HILARICS-2 in [8]. The authors of [1] specify a hypothetical processor using VHDL. They do this by describing everything down to the timing of the bus signals.

1.3 General aims of nML

In the following the main goals in the development of nML are presented together with the way of their realization.

1.3.1 Abstraction Level

The abstraction level nML aims at is that of the instruction set, i.e. the "programmer's model" of the processor. To program a machine, one needs

to know about

- the memory model,
- different kinds of registers,
- directly supported data types,
- the exact semantics of instructions (including "side-effects"),
- addressing modes,
- alignment restrictions,
- condition code usage and
- processor-internal data structures like pipes.

One doesn't necessarily have to know about "system" programming details like exceptions and interrupts. A machine description should be as precise as necessary, but not more. E.g., in writing a machine description for an assembler/disassembler, the semantics of instructions can be ignored at all.

nML is based on a minimal set of assumptions about the machine: a machine, when run, executes a *program* that is a series of *instructions*. A *program counter* (*PC*) points to the next-to-be-executed instruction while executing the current one. A machine has *state* stored in *memory locations*. The sole purpose of a program is to change the contents of these locations.

All that instructions do is changing the values of locations. There are no inter-instruction control flow constructs; the program flow is changed by writing to the PC location. Each instruction can be seen as a function from state to state. By composing the instructions, the semantics of the whole program can thusly be given.[2] For practical reasons, instruction semantics are not given as one function, but as a sequence of assignments of the basic form

$$location = function\ (\ location\ \ldots\ )$$

[2]It is not that simple because of the program counter. While a semantic function of type $State \rightarrow State$ can be given easily for any program, it is of no great value, because this function has to be applied iteratively until some halting condition is reached.

A finite set of primitive functions (arithmetic, shifting, masking) is assumed.

Traditionally, such assignment sequences are known as "register transfers". One early register transfer formalism was *ISP*. Variations of this formalism were used in retargetable code generation [3] and peephole optimization [4] systems.

1.3.2 Sharing in Descriptions

In complex architectures, there may be hundreds of different combinations of operations and addressing modes. If instructions or addressing modes have side effects (e.g., setting condition codes), the semantic description of a single complete instruction may grow quite large. One goal of in the development of **nML** was therefore the reduction of description size by sharing as much of descriptions as possible.

To achieve this goal, the instruction set is enumerated by an attributed grammar. E.g, a machine may have a dozen numeric instruction that share the behaviour of conditionally setting a 'zero flag'. This can be modelled by a grammar fragment

```
mem tmp_src[1,long]                \ temporary registers
mem tmp_dst[1,long]

\ grammar rule for binary numeric operations
\ SRC and DST are addressing modes, described elsewhere

op numeric_instruction(a:num_action,src:SRC,dst:DST)
action={
        tmp_src=src;
        tmp_dst=dst;
        a.action;                  \ execute the numeric_action
        if tmp_dst==0              \ is the result zero?
         then CZ=1;                \ yes: set zero flag
         else CZ=0;                \ no: clear it
        endif;
        dst=tmp_dst;
       }
```

```
op num_action= add | sub | ...

op add()
action={
        tmp_dst=tmp_dst+tmp_src;
       }
```

The semantic action of any instruction is composed of fragments that are distributed over the whole grammar tree. This has been compared to the "inheritance" of object-oriented languages; in the above example, `numeric_instruction` is an "abstract base class" for add, sub,..., providing all "shared behaviour" for the latter.

1.4 Restrictions of the Machine Model

No explicit provisions are made for the description of

- self-modifying code
- i/o devices
- interrupts
- the underlying operating system
- sub-instructions and multi-cycle-instructions.

These may be described 'by hand'. E.g., operation system calls may be modelled as instructions:

```
op openfd()
syntax="move #42,d0; trap 1"
action={canonical("fopen",a0,a1);}
```

Of course, much more elegant ways to describe these concepts may be added easily, but they would complicate the semantics a lot while not being very general. nML in its present form is a core language that can be extended at need.

2 Syntax and Semantics of the nML Attribute Grammar

A **nML** description is a file consisting of an attributed instruction grammar and assorted definitions.

2.1 General Description of Attribute Grammars

A *context-free grammar* G is a 4-tuple $G = (N, T, P, S)$ consisting of *nonterminals*, *terminals*, *production rules* and a *start symbol*. A *token* is a terminal or a nonterminal. The set S_G of *Strings* in G is the set of all sequences of tokens, i.e $S_G = (N_G \cup T_G)^*$. Traditionally, $\alpha, \beta, \ldots$ denote strings. N, T and P are finite; $N \cap T = \emptyset$, $S \in N$, $P \subset N \times S_G$.

A string t may be *derived in one step* from a string s, written $s \xrightarrow{1} t$ iff

$$s \equiv \alpha x \beta \ \wedge \ t \equiv \alpha \gamma \beta \ \wedge \ (x, \gamma) \in P$$

A string t may be *derived in n steps* from a string s, written $s \xrightarrow{n} t$, iff $\exists u : s \xrightarrow{1} u \ \wedge u \xrightarrow{n-1} t$. *$t$ is a derivation of s* ($s \xrightarrow{*} t$) if it may be derived in a finite number of steps.

A string α is *terminal* if it consists only of terminals, i.e. iff $\alpha \in T^*$. A grammar is *acyclic* if there exist no nonterminal x with a derivation $x \xrightarrow{*} \beta x \gamma, |\beta\gamma| > 0$.

The language $L(G)$ of a grammar G is the set $\{\alpha | \alpha \in T_G{}^* \wedge \ S_G \xrightarrow{*} \alpha\}$.

Acyclic grammars have finite languages.

An *attribute grammar* is a grammar in which for each nonterminal a fixed set of attributes, and for each production a set of *sematic rules* is given. For a given derivation, the semantic rules determine the values of the attributes. A theory of attribute grammars is given in [7]. [3]

[3] Full-fledged attribute grammars know two kinds of attributes: *synthesized* and *inherited*. If both occur together unrestricted, attribute evaluation can become quite expensive. For **nML** applications, synthesized attributes should suffice, so inherited attributes are silently ignored.

2.2 nML grammars

For nML grammars, all nonterminals have to have derivations. There may be no cycles. Together, this implies that all strings that have no productions are terminal.

A nML grammar description differentiates between two subsets of N, $N_\wedge$ and $N_\vee$, and two sets of production rules $P_\wedge$ and $P_\vee$. $N_\wedge \cup N_\vee = N, N_\wedge \cap N_\vee = \emptyset, P_\wedge \cup P_\vee = P, P_\wedge \cap P_\vee = \emptyset$.

$P_\wedge$ is the set of production functions $N_\wedge \rightarrow T_G$; while $P_\vee$ is the set of production relations $P_\vee \subset N_\vee \times N$. $P_\wedge$ models *and-rules*, while $P_\vee$ models *or-rules*. In an nML-grammar, there may be no 'mixed' rules.

Semantically, each terminal string produced by the grammar corresponds to one instruction in the instruction set. By itself, such a string contains no useful information: the instruction's syntax and semantics are defined by the string's attributes. For each attribute, its semantic function is given, i.e. the attribute declaration declares the attribute and its definition in one step.

Textually, a production in $P_\vee$ looks like

```
op n0 = n1 | n2 | n3 | ...
```

while a production in $P_\wedge$ with looks like

```
op n0(p1:t1,p2:t2, ... )
a1 = e1  a2 = e2 ...
```

where each n*i* is a nonterminal and each t*i* is a token. Each a*i* is an attribute name, the e*i* their respective definitions. The p*i* are just names for the parameters to be used in the attribute definitions.

Productions in $P_\vee$ have no attribute definitions; Nonterminals in $N_\vee$ simply pass the attributes through.

The start symbol is fixed to be the identifier `instruction`.

Attributes have *expressions* as their definitions. Expressions are arbitrary C-like expressions or sequences of statements. Expressions may contain references to attributes of parameters. An attribute reference *param.attr* refers to the value of the attribute *attr* of the paramter *param*.

What follows is a complete, if not very interesting, nML grammar:

```
op instruction(f:foo,g:bar)
```

```
size=f.size+g.size

op foo()
size=1

op bar()
size=2
```

this enumerates one 'instruction' , which has the one attribute `size` with value 3. In contrast, the grammar

```
op instruction(f:foo,g:barOrbaz)
size=f.size+g.size

op foo()
size=1

op barOrBaz = bar | baz

op bar()
size=2

op baz()
size=3
```

enumerates two 'instructions' of sizes 3 and 4.

3 The Pre-Defined Attribute Set

Three attributes are pre-defined: `syntax`, `image` and `action`.

The `syntax`-attribute describes the textual (=assembler) syntax of the instruction; it has to evaluate to a string.

The `image`-attribute describes the binary coding of the instruction, it has to evaluate to a *binary string*, which is a string containing only 1s, 0s and whitespace. The latter is ignored.

The `action`-attribute describes the semantics of an instruction; it has to evaluate to a sequence of register-transfer operations. The exact syntax of the latter is shown in section 4.

What follows is a short grammar that is complete in regard to the three pre-defined attributes.

```
type addr=card(24)
type long=card(32)
mem M[2^24,long]
mem PC[1,addr]

op instruction = jump | binop

op jump(a:addr)
syntax=format("jump %d",a)
image=format("1000 0000 %24b",a)
action={ PC=a; }

mem tmp1[1,long]
mem tmp2[1,long]

op binop(x:binaction,a1:addr,a2:addr)
syntax=format("%s %d,%d",x.syntax,a1,a2)
image=format("11%6b %24b %24b",x.image,a1,a1)
action= {tmp1=M[a1];
         tmp2=M[a2];
         x.action;
         M[a2]=tmp2;
```

```
}

op binaction= plus | move

op plus()
syntax="add"
image="00000"
action={ tmp2=tmp1+tmp2; }

op move()
syntax="move"
image="00001"
action={ tmp2=tmp1; }
```

This little grammar enumerates three instructions (or better: instruction templates).

The first of these has one terminal argument 'a' of type addr, i.e. `card(24)`. It describes the instruction `jump` x, where x may be any number between 0 and $2^{24} - 1$.[4] The semantics of the jump-instruction is described as the *sequence* containing the one assignment-statement `PC=a;`. The register `PC` has special semantics: it is assumed that during the execution of any instruction, `PC` holds the address of the instruction to be executed next. So, changing `PC` enables an instruction to jump somewhere else, where by default the instructions in a program are executed sequentially. This is, indeed, the only way to manipulate control flow.

The other two instructions, `add` and `move`, have a common description for fetching and storing arguments. In the case of the move operation, there is an 'unnecessary' load operation (`tmp2=M[a2]`), which is, however, semantically irrelevant. One can see how auxiliary registers are introduced to facilitate code sharing. As a side-effect, the physical data-path is re-modeled. These auxiliary registers don't have any impact on the functional model of the machine, because they cannot 'carry state' from one instruction to the next – at least not in the way they are used now.

[4] It is important to see that this grammar does *not* descibe $2^{24} + 2 * 2^{24-2}$ different instructions, but only three: to the grammar, all terminals/types are equally opaque.

3.1 Addressing Modes

nML supports the concept of addressing modes. Suppose the mode declarations

```
mem A[8,long]
mem D[8,long]

mode SRC = IMMS | IMMW | IMML | REG | IND | INDOFFSET

mode IMMS(n:int(8))=n
syntax=format("#%d",n)
image=format("%8b",n)

mode IMMW(n:int(16))=n
syntax=format("#%d",n)
image=format("%16b",n)

mode IMML(n:int(32))=n
syntax=format("#%d",n)
image=format("%32b",n)

mode REG= AREG | DREG

mode AREG(n:card(3))=A[n]
syntax=format("A%d",n)
image =format("%3b",n)

mode DREG(n:card(3))=D[n]
syntax=format("D%d",n)
image =format("%3b",n)

mode IND(R:AREG)=M[R]
syntax=format("(%s)",R.syntax)
image=R.image

mode INDOFFSET(R:AREG,O:DREG)=M[R+O]
```

```
syntax=format("(%s,%s)",0.syntax,R.syntax)
image=R.image
```

This is an incomplete subset of the addressing mode grammar of a 68000. (Incomplete, because no way exists to distinguish between modes. In reality, special marker bits would be provided by auxiliary attributes.)

One can see that the difference between addressing modes and 'normal' grammar rules ('op' rules) is the existence of a 'value'. E.g, the AREG rule has the value A[n], while the INDREG mode has the (composed) value M[A[n]]. Addressing modes are used as follows:

```
op add(src:REG,dst:REG)
action={ dst=src+dst; }
```

Now, if src is an AREG and dst is a DREG, this is the same as

```
action={ D[n]=A[m]+D[n]; }
```

for some values of n and m. That is, a parameter that stands for an addressing mode is replaced by its 'value'.

This can be modelled by a special attribute value. Simply imagine all mode declarations transformed into declarations like

```
...
op INDOFFSET(R:AREG,0:DREG)
value=M[R.value+0.value]
```

and all uses of modes into

```
...
action={ dst.value=src.value+dst.value; }
```

3.2 Type declarations

In addition to the instruction grammar, a nML description contains declarations for memory objects, data types, constants and macros.

A *data type* describes a set of values, e.g. the type card(8) describes the set of numbers 0...255. In the grammar, data types are used as terminals (they could as well be seen as grammar rules, i.e. card(8) could denote 256 expansions, but this would blow up the grammar without need). nML knows about the following type constructors:

- `int(`n`)`
  is the type of n-bit signed numbers in 2s-complement representation.
- `card(`n`)`
  is the type of n-bit unsigned numbers.
- `float(`n`,`m`)`
  is the type of floating-point numbers with n bit mantissa and m bit exponent. While no provision for NaNs and infinities are made, a IEEE-754 representation may be assumed.
- `fix(`n`,`m`)`
  is the type of signed fixed-point numbers with n bits before and m bits after the binary point.
- `[`n`,`m`]` (where $n \leq m$)
  is the type of (integer or cardinal) numbers in the range of $[n \ldots m]$.
- `enum(`$id_1, \ldots, id_i$`)`
  defines an enumeration type, i.e. the type card($\lceil \log_2(i) \rceil$) and the constants id_1=0, ..., id_i=$i-1$.
- `bool`
  denotes the boolean values. Two constants `true` and `false` are predefined. If coerced to an integer, `true` has the value `-1`, while `false` has the value `0`. Wherever an integer is needed (essentially, only in `if`-expressions), a `0` will be interpreted as `false`, while everything else will be interpreted as `true`.

A *type definition* like

```
type byte=card(8)
```

defines a synonym for a type expression.

3.3 Memory declarations

A *memory declaration* like

```
mem A[8,card(32)]
```

defines a *memory base*, i.e. a set of memory *locations* accessible under a name and an index. A location is a place where an value of a type may be stored. E.g., the above shown declaration introduces a memory base called `A` that contains 8 locations, denotable as `A[0]`...`A[7]`, which may be used to store numbers in the range of $0 \ldots 2^{32} - 1$. Memory bases and locations are *not* terminals of the grammar; in fact, they don't exist in the grammar at all, only in the `action` attribute definitions.

Memory declarations can have additional attributes:

```
mem M[2^32,byte] alignment=2
```

declares an alignment restriction (if these are supported; they are *not* part of the core language);

```
mem A[8,int(32)]
mem SP[1,card(32)] alias=A[7]
```

declares `SP` to be an alias of `A7`, i.e. both denote the same location, but they have different type interpretations;

```
mem PORT1[1,byte] volatile="port1" alias=M[0xffffff84]
```

declares a memory location to be "volatile", i.e. able to change at random. The value of the `volatile` attribute may hold additional information for the entity that reads the description.

There is no predefined attribute for marking memory bases as 'temporary' – in the sense of the registers `tmp1` and `tmp2` in the first example – , because this property can be deduced automatically.

One last attribute is the `program_memory` declaration. A declaration of the form

```
mem M[32000] program_memory
```

declares the memory base `M` to be the one holding programs. Per default, the largest memory base is assumed to hold the programs.

When modelling machines where memory is divided into different purpose parts, one can combine the `program_memory` and the `alias` attributes:

```
mem MEM[2^24,byte]
mem PROGMEM[2^20] program_memory alias=MEM[1024]
```

declares the `PROGMEM` as part if the `MEM` memory base, starting at the address 1024.

3.4 Constants and Global Parameters

A declaration like

```
let A=100
```

declares a global constant `A` to have the value 100. Such a constant might be used in every context its value could stand. Any constant may be defined only once.

Constants may be used to extend nML: Any information about a machine that can be given with a single number or string can easily be defined as a constant (with a default value, so that standard nMLdescriptions still work).

In core nML, there is just one such constant (or "global parameter").

This is the `pipeline_factor`. On machines with an instruction pipeline visible to the programmer, there are *delay slots* whenever a jump occurs. Usually, there is one such slot, but two are not unheard of. A declaration

```
let pipeline_factor=1
```

introduces one delay slot after each instruction that changes the program counter. The default value is 0.

3.5 Macros

A macro-definition like

```
macro max(A,B)= if (A)>(B) then A else B endif
```

defines a pseudo-function. Macros may not introduce circularities, neither direct nor indirect! They are of no further interest, because a simple syntactic expansion can remove the painlessly.

4 Attribute Expression Syntax and Semantics

An *expression* is a term that can be evaluated to a value. A value is either a logic value, a number, or a string. Expressions are used both to compute values of attributes and as parts of register transfer sequences in action attribute values. These really are two different uses of the same expression syntax and semantics; this dual use leads to restrictions in the set of expressions allowed as direct values of attributes.

4.1 Expressions

An expression is either

- a *constant* like 13 or `"add %s,%4d"`. Numeric constants may be written to base 2 or base 16, as in `0b00100010` and `0x12ab`,
- an *attribute reference* like `arg1.syntax`,
- a *memory location* like `PC` or `M[12]` or `A[D[x-1]+4]`, containing the name of a memory base and an arbitrary indexing expression (Location bases of size 1 can be accessed without an index. Examples are the program counter and single condition bits.),
- a *function call* like `a+b` or `format("%s",a.x)`,
- a *macro application* like `MAX(a,b)` where `MAX` is defined by a macro definition like
  `macro MAX(A,B) = if (A)>(B) then A else B endif`
  Such a macro application can be evaluated by replacing it textually by its definition in the 'obvious' way.
- a *conditional* like `if a>b then x else y endif`, which returns the value of the evaluated expression,
- a *switch expression* like

```
switch x {
 case 0: "load"
 case 1: "store"
 default:"move"
}
```

that evaluates to the one selected value (the selection has to be exhaustive!).

There is a list of predefined functions and operators:

- +,-
  these are the usual arithmetic functions. Applied to two numbers of type X, they return type X. In the case of Integer(N) or Cardinal(N) arguments, all functions are defined modulo 2^N. Applied to Integer(N) and Integer(M) arguments, the result is Integer(max(N,M)). The same applies for different-sized Cardinal arguments. In the case of Cardinal(M) and Integer(N) arguments, the result is of type Integer(max(N,M)). In the case of floating point or fixed point arguments, both argument types have to be the same.

- *,/,%
  these are the usual multiplication, division and remainder functions. In the case of Integer or Cardinal arguments, the same rules apply as for + and -. In the case of floating point or fixed point arguments, mixing with integers and cardinals is allowed, the result type being that of the float or fix argument.

- ^
  is the integer exponentiation function. The second argument has to be a constant.

- >,<,>=,<=,==,!=
  The standard numeric comparison functions. These may be applied to all kinds of numbers. They return a boolean value, which is equivalent to either -1 (true) or 0 (false).

- `<<, >>, &, |, xor`
  the binary shift and mask functions from C. These may be applied to Integer and Cardinal values, only. ^ was renamed `xor` so that ^ be free for the exponentiation function.

- `&&, ||`
  The logic functions from C. They accept locic values and integers (0 is false) and deliver logic values.

- `coerce(`*typename*`,`*value*`)`
  this function, when applied to any numeric value, delivers the "best approximation" of the value in the type to be coerced to. When coercing signed to unsigned, the behaviour is undefined on negative values. When coercing from floating or fixed point numbers to integer, everything behind the binary dot is cut off.

- `canonical(`*string*`,`*args...*`)`
  this function applies a function of unknown semantics. It may be used in `action` attribute definitions only. To give an example of its use: a machine that directly implements trigonometric functions will need a register transfer like
  `dst=canonical("sin",src)`
  It is assumed that the entity that reads the description knows what is meant by the canonical function. Canonical functions may only be used as 'objects' in semantic attributes; they *must not* be used in computing the attributes themselves!

- `format(`*format-string*`,`*args...*`)`
  This function is used to put together the string values of the `syntax` and `image` attributes. The format string is a variation of the `printf` format string well known from C. It may contain alphanumeric characters, blanks, tabs ('\t'), newlines ('\n'), and format directives of the form %*nC*, where *n* is an optional field size and *C* is one of the following characters:

  - `d`
    This takes an Integer or Cardinal argument from the argument list and formats it as a decimal number.

- `b`
  This takes a Cardinal argument and formats it as a binary number. It may also take a binary string, i.e. a string containing only 1s, 0s and (ignored) whitespace.
- `x`
  This takes a Cardinal argument and formats it as a sedecimal number.
- `s`
  This takes a string argument and incorporates it as a whole.

4.2 Sequences

The `action` attribute has *register-transfer sequences* as value. Such a sequence is built up from *statements*. Textually, a sequence is enclosed by braces ({ and }); each statement in a sequence is delimited by a semi-colon (;).

A statement is either

- an *assignment* like `a=b+c`, where the result of an arbitrary expression is assigned to a location.
- a *conditional statement*, which looks like a conditional expression, but which contains two sequences instead of two expressions,
- a *switch statement*, which looks like a switch expression, but which contains sequences instead of expressions.

Sequences may contain calls to canonical function. Expressions occuring in sequences may refer to locations.

4.3 Coercion rules for assignment statements

The only kind of action done by a register transfer sequence is that of assigning values to locations. nML provides coercing rules for assignments between locations of different types.

There is one main rule: assignment between locations of *equal size* is a direct, un-coerced operation. Assignment between locations of *different size* is either done with a coercion, if the types are compatible, or not allowed.

Assume the definitions

```
type byte=card(8)
type sbyte=int(8)
type long=card(32)
type slong=int(32)
type float32=float(24,8) \ 24 bit mant., 8 bit exp.
mem M[2^32,byte]
mem D[8,long]
mem F[8,float32]
```

in which five data types and three memory bases are defined. Let's look at some statements using these definitions:

```
M[100]=M[101];
```

This simplest possible case moves a 'byte' from one location to another. No coercion or casting takes place.

```
D[0]=F[0];
```

Here, a 'float32' value is moved into a location that is tagged as 'long'. Since both locations have the same size (32 bits), the value is moved regardless of the incompatibility of types.

```
D[0]=M[100];
```

Here, a 'byte' is taken and put into a 'long' register. An implicit coercion takes place, i.e. what really happens is:

```
D[0]=coerce(long,M[100]);
```

In the case of coercion between signed and unsigned values, as in

```
mem SB[1,sbyte]
 ...
D[0]=SB[0];
```

presumably sign-extension is done. This is *not* guaranteed! To have guaranteed sign extension, use

```
mem SB[1,sbyte]
 ...
D[0]=coerce(slong,SB[0]);
```

Here, a signed byte is coerced (=extended) to a signed long; then the "equal size" rule takes charge and puts the value unchanged into the unsigned location.

Lastly, something like

```
F[0]=M[100];
```

is not allowed.

4.4 Assignment to Sequences of Locations

On most machines, addressing is per byte, while registers hold multi-byte values. Under declarations

```
mem M[2^32,byte]
mem D[16,long]
\begin{verbatim}
an assignment like
\begin{verbatim}
M[100]=D[0];
```

means to store the (long) value of D0 into the sequence of byte locations M[100]... M[103]. While the above sequence is *not* allowed in pure nML, a simple extension could be defined as follows: when a global constant `byte_order` is set to one of the strings `"big"` or `"little"`, the above statement is defined when the size of the destination is a multiple of the size of the source (It is still ill-defined to store a 30-bit value into a byte array). The semantics is that the source value is split up and distributed over the indices (in this case, 100... 103). The order is 'big-endian' (most significant byte first), if byte_order is set to "big", and 'little endian' otherwise.

A first problem arises: The semantics of

```
D[0]=M[100]
```

should now be changed in a symmetrical way to be that of

```
D[0]=M[100..103]
```

(which is, of course, unsyntactical). To load a byte into a long location under this changed semantics, a temporary must be introduced:

```
mem TMPBYTE[1,byte]
TMPBYTE=M[100];
D[0]=TMPBYTE;
```

A second problem is that of bounds: lets assume an assignment to the top of memory:

```
M[2^32-2]=D[0]
```

On most machines, this will 'swap over' to addresses 0 and 1. What happens if the memory has a size different then 2^N? Assume

```
mem X[1200]
X[1198]=D[0]
```

This could cause a trap or wrap around into some totally unexpected place.

A third problem is that of alignment: on aligned machines, the machine description has to provide the information that an instruction like

```
M[1]=D[0]
```

will cause an alignment exception. This information can be given as a memory attribute like

```
mem M[2^32,byte] alignment=2
```

Or, when different sizes have different alignments:

```
mem M[2^32,byte] alignment=true
```

which could define that values of size 2^{N*8} are aligned on addresses that have the last $N-1$ bit cleared. The latter would, as a side effect, solve the problem of wrap around at the end of memory (when the memory is of an aligned size, but any architecture missing *this* constraint would be *truly* weird!).

As said before: multi-word memory access is *not* part of the core nML language.

5 A complete nML description

In the following, a complete nML description of a simple RISC-like machine is given. While being quite small, additional complexity is introduced through complex addressing-modes.

```
\ small.m --- description of a small, fictional machine

let REGS=4              \ 2^4 registers

type word=card(16)
type long=card(32)
type index=card(REGS)   \ register index type

mem M[2^32,long]        \ main memory
mem R[2^REGS,long]      \ registers

mem CZ[1,bool]          \ condition code
mem CN[1,bool]          \ bits

mem PC[1,long]          \ program counter

\ 2 kinds of addressing modes: short (5 bit) and long (7 bit)
\ short:
\ name Image    Syntax
\ MEM  0nnnn    (Rn)
\ REG  1nnnn     Rn
\ long:
\      00<short>
\ IMM  1iiiiii   #x    ,in the range -32...31
\ INC  010nnnn  (Rn)+
\ DEC  011nnnn  -(Rn)
\ post-increment and pre-decrement are modelled by attributes

mode MEM(i:index)=M[R[i]]
syntax=format("(R%d)",i)
```

```
image=format("0%4b",i)

mode REG(i:index)=R[i]
syntax=format("R%d",i)
image=format("1%4b",i)

mode SHORT = MEM | REG

mode LSHORT(s:SHORT) = s
syntax=s.syntax
image=format("00%b",s.image)
pre={}        \ these dummies have to be inserted to
post={}       \ make the attributes defined for all LONGs

mode IMM(n:int(6))=n
syntax=format("#%d",n)
image=format("1%6b",n)
pre={}
post={}

mode PRE(r:MEM)=r
syntax=format("-%s",r.syntax)
image=format("010%4b",r.image&0b1111) \ remove tag bit
  \ the removal uses the fact that bit strings are just
  \ numbers with a field size, so arithmetic operations
  \ like masking can be used on them
pre={ r=r-1; }
post={ }

mode POST(r:MEM)=r
syntax=format("%s+",r.syntax)
image=format("011%4b",r.image&0b1111) \ remove tag bit
pre={ }
post={ r=r+1; }

mode LONG = LSHORT | IMM | PRE | POST
```

```
op instruction(x:instr_action)
action={            \ these are the actions done in
                    \ each instruction
        R[0]=0;     \ R0 holds 0 constantly
        x.action;   \ here the different actions are inserted
       }
syntax=x.syntax
image=x.image

op instr_action = control_op | alu_op | move_op

op control_op = test_op | branch_op
              | jsr_op | rts_op

op test_op(src1:LONG,src2:SHORT)
action={
        src1.pre;
        CZ=src1==src2;
        CN=src1<src2;
        src1.post;
       }
syntax=format("cmp %s,%s",src1.syntax,src2.syntax)
image =format("0000 %b %b",src1.image,src2.image)

type testcode = enum(tr,          \ true
                     zc,zs,       \ CZ clr/set
                     nc,ns)       \ CN clr/set

op branch_op(newpc:LONG,code:testcode)
action={
        newpc.pre;
        if code==tr
        ||(code==zc && CZ==0)
        ||(code==zs && CZ!=0)
        ||(code==nc && CN==0)
```

```
        ||(code==ns && CN!=0)
        then PC=newpc;
        endif;
        newpc.post;
       }
syntax=format("b%s (%s)",switch(code){
                                  case tr: "ra"
                                  case zc: "eq"
                                  case zs: "ne"
                                  case nc: "mi"
                                  case ns: "pl"
                                 },newpc.syntax)
image =format("0001 0%3b %b",code,newpc.image)

op jsr_op(nextpc:LONG,link:SHORT)
action={
        nextpc.pre;
        link=PC;
        PC=nextpc;
        nextpc.post;
       }
syntax=format("jsr (%s),%s",nextpc.syntax,link.syntax)
image =format("0110 %b %b",nextpc.image,link.image)

op rts_op(link:LONG)
action={
        link.pre;
        PC=link;
        link.post;
       }
syntax=format("rts (%s)",link.syntax)
image =format("0111 %b",link.image)

mem SRC1[1,long]        \ temporary registers
mem SRC2[1,long]
mem DST[1,long]
```

```
op alu_op(src:LONG,dst:SHORT,aa:alu_action)
action={
        src.pre;
        SRC1=src;
        SRC2=dst;
        aa.action;
        dst=DST;
        src.post;
       }
syntax=format("%s %s,%s",aa.syntax,src.syntax,dst.syntax)
image =format("1%b %b %b",aa.image,src.image,dst.image)

op alu_action= a_add | a_sub | a_and | a_or | a_mult | a_div | a_rem

op a_add()
action={ DST = SRC1 + SRC2; }
syntax="add"
image="000"

op a_sub()
action={ DST = SRC1 - SRC2; }
syntax="sub"
image="001"

op a_and()
action={ DST = SRC1 & SRC2; }
syntax="and"
image="010"

op a_or()
action={ DST = SRC1 | SRC2; }
syntax="or"
image="011"

op a_mult()
```

```
action={ DST = SRC1 * SRC2; }
syntax="mult"
image="100"

op a_div()
action={ DST = SRC1 / SRC2; }
syntax="div"
image="101"

op a_rem()
action={ DST = SRC1 % SRC2; }
syntax="rem"
image="110"

op move_op = move1 | move2 | store | lconst | sconst

op move1(src:LONG,dst:SHORT)
action={
        dst=src;
       }
syntax=format("move %s,%s",src.syntax,dst.syntax)
image =format("0010 %b %b",dst.image,src.image)

op store(src:SHORT,dst:LONG)
action={
        dst=src;
       }
syntax=format("move %s,%s",src.syntax,dst.syntax)
image =format("0011 %b %b",src.image,dst.image)


op lconst(dst:REG,value:long)    \ the only >1-word-instruction
action={
        dst=value;
       }
syntax=format("move #%d,%s",value,dst.syntax)
```

```
image =format("0100 %b 0000000 %b",dst.image,value)

op sconst(dst:SHORT,value:int(7))
action={
        dst=coerce(int(32),value);
       }
syntax=format("moveq #%d,%s",value,dst.syntax)
image =format("0101 %b %b",dst.image,value)
```

6 Appendix: Grammar of nML

What follows is a kind of typed EBNF grammar. An annotation like foo_{bar} means that "*foo*" is of type "*bar*". $(X)^*$ means "a sequence of 0 or more Xs". $X \mid Y$ means "either X or Y". $[X]$ means "one X or nothing".

machine-description ⇒
    (*memory-spec*
    | *type-spec*
    | *mode-spec*
    | *op-rule*
    | *let-def*
    | *macro-def*)*

memory-spec ⇒
    mem $name_{mem\ type}$ [$expr_{card}$, $name_{type}$] (*mem-attribute*)*

mem-attribute ⇒
    volatile = $expr_{string}$
    | alias = *location*

type-spec ⇒
    type $name_{typespec}$ = $expr_{typespec}$

$expr_{typespec}$ ⇒
    bool
    | int($expr_{card}$)
    | card($expr_{card}$)
    | fix($expr_{card}$, $expr_{card}$)
    | float($expr_{card}$, $expr_{card}$)
    | [$expr_{int}$ .. $expr_{int}$]
    | enum(i_{card}...i_{card})

$expr_{type}$ ⇒
    $const_{type}$
    | $location_{type}$
    | $function\text{-}application_{type}$
    | if $expr_{bool}$ then $expr_{type}$ else $expr_{type}$ endif

| switch($expr_{select-type}$)
{
((case $const_{select-type}$ | default) : $expr_{type}$)*
}

function-application $_{type}$ ⇒
$name_{type_1,\ldots,type_n \rightarrow type}$ ($expr_{type_1}$, ... , $expr_{type_n}$)
| $expr_{type_1}$ $name_{type_1,type_2 \rightarrow type}$ $expr_{type_2}$

location $_{type}$ ⇒
$name_{mem\ type}$ [$expr_{card}$]
| $name_{mem\ type}$ [5]

$name_{int,int \rightarrow int}$ ⇒
+ | - | * | div | mod

$name_{card,card \rightarrow card}$ ⇒
& | | | ^ | >> | << | xor

$name_{bool,bool \rightarrow bool}$ ⇒
&& | ||

$name_{type,type \rightarrow bool}$ ⇒
>= | > | <= | < | == | !=

$name_{format-string,type,\ldots,type \rightarrow string}$ ⇒
format

$name_{type-specifier,type \rightarrow specified\ type}$ ⇒
coerce

op-rule ⇒
and-rule
| *or-rule*

or-rule ⇒
op $name_{op}$ = id_{op} | ... | id_{op}

[5]This abbreviation ($x \equiv x[0]$) is only valid if x denotes a memory base of size 1.

and-rule ⇒
    op *name*$_{op}$ (*id*$_{param}$: *name*$_{typespec}$, . . . , *id*$_{param}$: *name*$_{typespec}$)[6]
        (*attribute-def*)*

mode-spec ⇒
    mode-and-rule
    | *or-rule*

mode-and-rule ⇒
    op *name*$_{op}$ (*id*$_{param}$: *name*$_{typespec}$, . . . , *id*$_{param}$: *name*$_{typespec}$) [=*expr*]
        (*attribute-def*)*

attribute-def ⇒
    name$_{attr}$ = (*expr* | *sequence*)

sequence ⇒
    (*statement* ;)*

statement ⇒
    location$_{type_1}$ = *expr*$_{type_2}$
    | if *expr*$_{bool}$ then *sequence* [else *sequence*] endif
    | switch(*expr*$_{select-type}$)
      {
        ((case *const*$_{select-type}$ | default) : *actions*)*
      }

let-def ⇒
    let *name*$_{type}$ = *expr*$_{type}$

macro-def ⇒
    macro *name*$_{macro}$(*param* , . . . , *param*) = *expr*

[6]The parameter list may have length 0.

7 Appendix: Selected Problems

This appendix shall show how concepts like data pipelining and interrupts, which are not directly supported, can be modelled by adding constant preambles and postludes to each instruction.

7.1 Pipelines

Consider a machine with a 3-cycle multiplication. How may this be modelled? Lets assume a 16 bits * 16 bits multiply with a 32-bit result in a register pair. This shall be written down as

```
mult D1,D2,D3
```

and have the semantics "compute D1*D2 and put the result into the register-pair D3/D4". The storing of the result shall occur 3 cycles later, i.e. in the program

```
move #1,D3
move #2,D4
mult D1,D2,D3      \ start multiplication...
add  #1,D3         \
move D3,D1         \ D1=1
move D4,D2         \ D2=2
\ now the multiplication results are transferred to D3,D4
move D3,D0         \ move the high word of D1*D2 to D0
```

So two tasks have to be modelled: first the computing over a time of 3 cycles, and then the storing. Assumed that the multiplication is fully pipelined, one could write something like

```
mem D[16,word]              \ a few registers

mem mult_dst0[1,long]
mem mult_dst1[1,long]
mem mult_dst2[1,long]
mem mult_dst3[1,long]
```

```
mem mult_flag0[1,long]
mem mult_flag1[1,long]
mem mult_flag2[1,long]
mem mult_flag3[1,long]

mem mult_output0[1,card(4)]
mem mult_output1[1,card(4)]
mem mult_output2[1,card(4)]
mem mult_output4[1,card(4)]

op instruction(i:rest_instruction) \ root of the instruction tree
action={
        mult_flag0=0;              \ no multiplication started
        i.action;
        if mult_flag3 then
          D[mult_dst3]=mult_output3 >>16;     \ high word
          D[mult_dst3]=mult_output3 & 0xffff; \ low word
        endif;
        mult_output3= mult_output2; \ advance pipeline
        mult_dst3   = mult_dst2;
        mult_flag3  = mult_flag2;
        mult_output2= mult_output1;
        mult_dst2   = mult_dst1;
        mult_flag2  = mult_flag1;
        mult_output1= mult_output0;
        mult_dst1   = mult_dst0;
        mult_flag1  = mult_flag0;
       }

op rest_instruction = ... | mult | ...  \ many different operations,
                                         \ amongst them mult
op mult(x:card(4),y:card(4),dst:card(3))
action={
        mult_output0=D[x]*D[y];
        mult_dst0=dst;
        mult_flag0=true;
```

```
}
```

How does it work? The pipeline is modelled by 3 sets of registers: one modelling the data part of the pipeline, the second one remembers the destination register, the third one flags whether a multiplication is going on at all. The 'mult' operation multiplies the contents of the registers, inserts the result into the first step of the pipeline, stores the destination register, and sets the flag. Now, at the beginning of *each* instruction, the multiplication pipeline is advanced one step (regardless of wether it is filled or not). So, the cycles go as follows:

```
   mult D1,D2,D3  mult_output0=D1*D2;
-------------------------------------------
                  mult_output1=mult_output0
   add  #1,D3
-------------------------------------------
                  mult_output2=mult_output1
   move D3,D1
-------------------------------------------
                  mult_output3=mult_output2
   move D4,D2
                  D3,D4=mult_output3
-------------------------------------------
   move D3,D0
```

How does one model a not-pipelined multi-cycle operation? One way is to use a counter instead of a pipeline structure:

```
mem D[16,word]

mem mult_reg[1,long]
mem mult_cnt[1,card(3)]
mem mult_dst[1,card(4)]

op instruction(i:rest_instruction)
action={
        i.action;
```

```
        if mult_cnt==1 then
          D[mult_dst]=mult_reg >> 16; \ high word
          D[mult_dst]=mult_reg & 0xffff; \ low word
        endif;
        if mult_cnt>0
        then mult_cnt=mult_cnt-1
        endif;
       }

op rest_instruction = ... | mult | ...

op mult(x:card(4),y:card(4),dst:card(3))
action={
        mult_cnt=4;
        mult_reg=x*y;
        mult_dst=dst;
       }
```

Here, a mult_cnt of 0 marks an inactive pipeline. If the pipeline is active and the count is on 1, the value is stored.

7.2 Interrupts

One can model interrupts in about the same way as pipelines. Assume an interrupt register that may hold a value of 0 or an interrupt number that serves as index into some vector array stored at address 256.

```
mem interrupt_register[1,card(4)] volatile="irq"

op instruction(i:rest_instruction)
action={
        i.action;
        if interrupt_register!=0
        then STORED_PC=PC;
             PC=M[interrupt_register<<2+0x100];
             interrupt_register=0;
        endif;
```

```
}
```

The interrupt-register is marked as "volatile", i.e. "changing its value". If some non-0 value appears, the PC is stored in some intermediate location (or put on the stack or whatever) and changed to the address found at the index. Of course, on a real machine much more happens: the current CPU state is stored, special mode bits are set, interrupts may be masked, etc.

References

[1] Peter J. Ashenden:
The VHDL Cookbook
First Edition, July 1990
Dept. Computer Science, Univ. of Adelaide, South Australia

[2] CAD Language Systems Inc:
VHDL Language Reference Manual
Draft Standard 1076/A, 31 December 1986

[3] R.G.G. Cattell:
Automatic Derivation of Code Generators from Machine Descriptions, in: ACM Transactions on Programming Languages and Systems 2(2), April 1980, pp. 173-190

[4] J.W. Davidson and C.W. Fraser:
The Design and Application of a Retargetable Peephole Optimizer, in: ACM Transactions on Programming Languages and Systems 2(2), April 1980, pp. 191-202

[5] Edif Steering Committee:
EDIF Specification Version 1.1.1
June 1986

[6] Edif Steering Committee:
EDIF Electronic Design Interchange Format Version 2.0.0 Draft
December 1986

[7] Gilberto File:
Theory of Attribute Grammars
Ph.D. thesis, Technische Hogeschool Twente, 1983

[8] Robert Severyns, Eric Willems:
HILARICS-2: The Language
December 7, 1989, Preliminary Release V1.0

[9] Richard M. Stallman:
Using and Porting GNU CC
version tagged as "last updated 12 September 1989, for version 1.36"

07/31/2002 09/246,047

31jul02 13:58:58 User267149 Session D251.1

SYSTEM:OS - DIALOG OneSearch
File 348:EUROPEAN PATENTS 1978-2002/Jul W03
(c) 2002 European Patent Office
File 349:PCT FULLTEXT 1983-2002/UB=20020725,UT=20020718
(c) 2002 WIPO/Univentio

Set	Items	Description
S1	2297	(CONFIGUR??????(3N)PROCESSOR? ?)/TI,AB,CM
S2	192	((DESCRIPT?????? OR DESCRIB??????)(3N)HARDWARE)/TI,AB,CM
S3	738	(HARDWARE(3N)IMPLEMENT??????)/TI,AB,CM
S4	1081	(IMPLEMENT??????(3N)PROCESSOR? ?)/TI,AB,CM
S5	471	HDL/TI,AB,CM
S6	14	(SYNTHES??????(3N)SCRIPT??????)/TI,AB,CM
S7	15	((PLACE OR ROUTE)(3N)SCRIPT??????)/TI,AB,CM
S8	118	(TEST? ?(3N)BENCH? ?)/TI,AB,CM
S9	82	(CONFIGUR??????(3N)SPECIFICAT??????)/TI,AB,CM
S10	74	(SOFTWARE(3N)DEVELOP??????(3N)TOOL? ?)/TI,AB,CM
S11	416	(SOFTWARE(3N)DEVELOP??????)/TI,AB,CM
S12	318	(SOFTWARE(3N)TOOL? ?)/TI,AB,CM
S13	11277	(COMPILER? ? OR ASSEMBLER? ? OR LINKER? ? OR DISASSEMBLER? ? OR DEBUGGER? ?)/TI,AB,CM
S14	15	(INSTRUCTION(3N)SET(3N)SIMULATOR? ?)/TI,AB,CM
S15	1431	(DIAGNOSTIC? ?(3N)TEST? ?)/TI,AB,CM
S16	268	(TEST??????(3N)TOOL? ?)/TI,AB,CM
S17	250	((ENHANC? OR BETTER OR OPTIMUM OR OPTIMAL OR OPTIMIS?????? OR OPTIMIZ??????)(3N)IMPLEMENT??????)/TI,AB,CM
S18	2316	(AUTOMAT??????(3N)PROCESS? ?)/TI,AB,CM
S19	70	((FAST?????? OR EXPEDIT??????)(3N)DEVELOP??????)/TI,AB,CM
S20	663	S10:S12
S21	1932	S2:S4
S22	112	S1 AND S21
S23	4	S22 AND S20
S24	4	IDPAT (sorted in duplicate/non-duplicate order)
S25	4	IDPAT (primary/non-duplicate records only)
S26	108	S22 NOT S25
S27	1	S26 AND S5
S28	107	S26 NOT S27
S29	60	(HARDWARE()DESCRIPT?????()LANGUAGE? ?)/TI,AB,CM
S30	510	S5,S29
S31	2	S28 AND S30
S32	2	S31 NOT S27
S33	105	S28 NOT S32
S34	0	S33 AND (S6 OR S7 OR S8)
S35	1	S33 AND S9
S36	104	S33 NOT S35
S37	0	S36 AND S20
S38	5	S36 AND S13
S39	5	IDPAT (sorted in duplicate/non-duplicate order)
S40	5	IDPAT (primary/non-duplicate records only)
S41	103	S26 NOT S40
S42	0	S41 AND S15
S43	0	S41 AND S16
S44	4	S41 AND S17
S45	99	S41 NOT S44
S46	1	S45 AND S18
S47	12	S1 AND S2
S48	6	S47 AND S3
S49	6	IDPAT (sorted in duplicate/non-duplicate order)
S50	6	IDPAT (primary/non-duplicate records only)

S51	6	S47 NOT S50
S52	6	IDPAT (sorted in duplicate/non-duplicate order)
S53	6	IDPAT (primary/non-duplicate records only)
S54	24	S1 AND S3
S55	18	S54 NOT S47
S56	2	S55 AND (CONFIGUR??????(6N)SPECIFICAT??????)
S57	16	S55 NOT S56
S58	2	S57 AND S4
S59	2	S58 NOT S56
S60	92	S45 NOT S47
S61	14	S57 NOT S58
S62	93	S60,S61
S63	0	S62 AND S10
S64	0	S62 AND S20

25/TI,PN,PD,PY,K/1 (Item 1 from file: 349)

ANONYMOUS TRANSACTION SYSTEM
SYSTEME DE TRANSACTION ANONYME
Patent and Priority Information (Country, Number, Date):
Patent: WO 200242982 A2 20020530 (WO 0242982)
Publication Year: 2002

.. The intermediary allows suppliers of personal financial products and services 560 access to the consumer's anonymous financial profile 550, and in a preferred embodiment **software tools** are provided so that the suppliers 560 can group and sort consumers according to the information in the consuniers' anonymous profiles 5 5 0, which...the functional blocks and software modules or features of the flexible interface call be implemented by themselves, or in combination with other operations in either **hardware** or software. Having **described** and illustrated the principles of the invention in a preferred embodiment thereof, it should be apparent that the invention can be modified in arrangement and...offers for goods and services with better terms than offers that the client has already accepted.

29 An anonymous transaction system, comprising:
one or more **processors configured** to receive first party data and convert the
first party data into an anonymous profile;
51
at least one of the **processors configured** to receive criteria from a second party
for providing a transaction; and
at least one of the **processors configured** to initiate the transaction when the anonymous profile meets the criteria.

30 An anonymous transaction system according to claim 29 wherein at least one of...

25/TI,PN,PD,PY,K/2 (Item 2 from file: 349)

AUTOMATED PROCESSOR GENERATION SYSTEM FOR DESIGNING A **CONFIGURABLE PROCESSOR** AND METHOD FOR THE SAME
SYSTEME AUTOMATISE DE PRODUCTION DE PROCESSEURS, DESTINE A LA CONCEPTION D'UN PROCESSEUR CONFIGURABLE, ET PROCEDE ASSOCIE
Patent and Priority Information (Country, Number, Date):
Patent: WO 200161576 A2 20010823 (WO 0161576)
Publication Year: 2001

1. A system for designing a **configurable processor**, the system comprising: hardware generation means for, based on a configuration specification including a predetermined portion and a user-defined portion, generating a **description** of a **hardware implementation** of the **processor**; and
software generation means for, based on the configuration specification, generating **software development tools** specific to the **hardware implementation**;
wherein the **hardware** generation means is for, based on the user-edp on o configuration specification, including a user-defined register file in the **description** of the **hardware implementation** of the **processor**; and
the software generation.means is for, based on the user-defined portion, including software related to the user-defined processor register file in the **software development tool**
3 The system of claim 2, wherein the hardware generation means is for generating at least part of the description of the **hardware implementation** in a register transfer level **hardware description** language.

8 The system of claim 1, wherein the hardware generation means is for generating, as part of the **processor hardware implementation description**, a **description** of logic to assign write ports of the user-defined register file to instruction operands to minimize data staging costs.

12 The system of claim 1, wherein the hardware generation means is for generating, as part of the **hardware implementation** of the **processor**, logic to provide a read port for the register file for each field, within an instruction accessing the register file, used to select a source operand from the register file.

15 The system of claim 1, wherein the hardware generation means is for generating, as part of the **hardware implementation** of the **processor**, interlock logic for accessing the register file.

39 A system for designing a **configurable processor**, the system comprising: hardware generation means for, based on a. configuration specification including a predetermined portion and a user-defined portion, generating a **description** of a **hardware implementation** of the **processor**; and
software generation means for, based on the configuration specification,

generating **software development tools** specific to the **hardware implementation;**
wherein the configuration specification includes a statement specifying scheduling information
of instructions used in the **software development tools;**
the hardware generation means is for, based on the configuration specification, generating a description of at least one of pipeline logic, pipeline stalling logic and...

25/TI,PN,PD,PY,K/3 (Item 3 from file: 349)

AUTOMATED PROCESSOR GENERATION SYSTEM FOR DESIGNING A **CONFIGURABLE PROCESSOR** AND METHOD FOR THE SAME
SYSTEME DE GENERATION D'UN PROCESSEUR AUTOMATISE DESTINE A LA CONCEPTION D'UN PROCESSEUR CONFIGURABLE ET PROCEDE CONNEXE
Patent and Priority Information (Country, Number, Date):
Patent: WO 200046704 A2 20000810 (WO 0046704)
Publication Year: 2000

A **configurable** RISC **processor implements** a user-definable instruction set with high performance fixed and variable length encoding. The process of defining new instruction sets is supported by tools that...

...them, to maintain multiple instruction sets & to easily switch between them. A standardized language is used to develop configurable definitions of target instructions sets, HDL **descriptions** of **hardware** needed to **implement** the instruction set, and development tools for verification and application development, thus enabling a high degree of automation in the design process.

Claim

1. A system for designing a **configurable processor**, the system comprising: means for, based on a configuration specification, generating a **description** of a **hardware implementation** of the **processor**; and
means for, based on the configuration specification, generating **software development tools** specific to the **hardware implementation**.

2 The system of claim 1, wherein the means for generating **software development tools** comprises means for generating **software development tools** capable of generating code to run on the processor.

13 The system of claim 1, wherein the **hardware implementation description** includes at least one of a detailed HDL **hardware implementation description**; synthesis scripts; place and route scripts; programmable logic device scripts; a test bench; diagnostic test for verification; scripts for running diagnostic tests on a simulator; and test tools.

14 The system of claim 1, wherein the means for generating the **hardware implementation description** comprises:
means for generating a **hardware description** language **description** of the **hardware implementation description** from the configuration specification;
means for synthesizing logic for the **hardware** implementation based on the **hardware description** language **description**; and

25/TI,PN,PD,PY,K/4 (Item 4 from file: 349)

SYSTEM FOR DIVIDING PROCESSING TASKS INTO SIGNAL PROCESSOR AND DECISION-MAKING MICROPROCESSOR INTERFACING
SYSTEME DE SEPARATION DES TACHES DE TRAITEMENT EN TACHES POUR INTERFACAGE AVEC UN PROCESSEUR DE SIGNAUX ET UN MICROPROCESSEUR DE PRISE DE DECISION
Patent and Priority Information (Country, Number, Date):
Patent: WO 9308524 A1 19930429
Publication Year: 1993
.. an environment which accounts for and provides connection and interfaces with the host microprocessor. It is another object of the invention to provide a programmable, **configurable**, real time signal **processor** which is particularly suited to the requirements of signal processing and which conducts deterministic real time signal processing and interfaces with a microprocessor which conducts...SPROC can be individually set or cleared via the parallel port, as described below. GP[3:01 are four general purpose pins that are individually **configurable** as either inputs or outputs. During reset, when RESET is LOW, all GPIO signals are set up as inputs. In addition to being subject to internal program control, the configuration of each GP pin, and the value of each GPIO signal **configured** as an output, are also individually controllable via the parallel port.

B. SPROC **Development** System and **Software**

'fhe above-disclosed SPROC devices 10 are preferably programmed via a development system (SPROCIab). The SPROCIab development system is a complete set of hardware and **software tools** for use with a PC to create, test, and debug digital signal processing designs. It was created as a design tool to support the development...

.cable connects the SPROCbox interface unit to the SPROCboard evaluation board. A security key connects to the PC parallel port. It enables use of the **development** system **software**. An integral power cord connects the power supply unit to the AC outlet. A positive-locking DC power cable connects the power supply to the SPROCbox 'interface unit. An auxiliary DC power cable daisy chains power from the interface unit to the SPROCboard evaluation board. The **software** components of the **development** system are described as follows: The SPROCIab development system shell executes under MS-DOS and provides access to all **development** system **software** components from a selection menu. The shell controls function calls among **development** system **software** components and provides a means for the designer to change certain system defaults. The SPROCview graphical design interface provides for easy creation of signal flow...OSC, STEOLIN9 STEO - OUT, SUM2 through SUM10, TRANSFNC, UCOMPRES, UEXPAND, VC0LSQR, and VOLTREF.

B.2 Entering a Diagram

The SPROC[ink microprocessor interface (SMI) provides **software** components necessary to **develop** microprocessor applications in ANSI C that include the SPROC chip as a memorymapped device. Using the development system the designer captures the signal processing subsystem ...on the signal flow block diagram. In fact, the user could view all of the signals that are internal to each biquad. In the current

hardware implementation of' the software-directed probe, values accessed using the probe command are made available as output from an on-chip, 8-bit, digital-to-analog...blocks. Code for signal processing functions is written at the primitive level. These primitive blocks comprise the SPROCcells function library, They are optimized for the **hardware** and efficiently **implemented** to extract maximum performance from the SPROC chip. Other primitive blocks include the glue blocks or phantoms required to provide. control and synchronization functions for...correctly access and interpret the 24-bit fixed-point data type native to the SPROC chip. The components of SMI are not like other SPROCIab **software tools**; they are not invoked from the **development** system environment. Instead, the components of SMI provide source code, in ANSI C. that is used outside of the SPROCIab development system, in ...micro keyword are available to the microprocessor.

27/TI,PN,PY,PD,K/1 (Item 1 from file: 349)

METHOD AND APPARATUS FOR MANAGING THE CONFIGURATION AND FUNCTIONALITY OF A SEMICONDUCTOR DESIGN
METHODE ET APPAREIL PERMETTANT DE GERER LA CONFIGURATION ET LA FONCTIONNALITE D'UN MODELE DE SEMI-CONDUCTEUR

Patent and Priority Information (Country, Number, Date):
Patent: WO 200022553 A2 20000420 (WO 0022553)
Publication Year: 2000

A method of managing the configuration, design parameters, and functionality of an integrated circuit (IC) design using a **hardware description** language (**HDL**). Instructions can be added, subtracted, or generated by the designer interactively during the design process, and customized **HDL** descriptions of the IC design are generated through the use of scripts based on the user-edited instruction set and inputs. The customized **HDL** description can then be used as the basis for generating "makefiles" for purposes of simulation and/or logic level synthesis. The method further affords the ability to generate an **HDL** model of a complete device, such as a microprocessor or DSP. A computer program implementing the aforementioned method and a hardware system for running the...

French Abstract
...configuration, de parametres de conception et de la fonctionnalite d'un modele de circuit integre (CI) a l'aide d'un langage de conception materielle (**HDL**). Il est possible d'ajouter des instructions, de les retrancher ou de les produire a l'aide du concepteur et ce, de maniere interactive durant le processus de conception, ainsi que de produire des descriptions **HDL** personnalisees du modele de CI a l'aide de scripts reposant sur un jeu d'instruction et des donnees d'entree edites par l'utilisateur. La description **HDL** personnalisee peut ensuite servir de base a la production de faconneurs de fichiers a des fins de simulation et/ou de synthese de niveau logique. Cette methode donne egalement la possibilite de produire un modele **HDL** d'un dispositif complet, un microprocesseur ou un DSP. L'invention concerne egalement un programme informatique permettant de mettre en oeuvre la methode susmentionnee ainsi...

Claim
... to create a customized description language model; and synthesizing said design based on said description language model.

2 The method of Claim 1, wherein said **description** language comprises a **hardware description** language (**HDL**
.**language model of said integrated circuit design; generating a netlist which is descriptive of the circuitry of said integrated circuit; compiling said netlist and said hardware description** model to produce a compiled integrated circuit design; I 0 fabricating at least one mask representing said compiled integrated circuit design; and fabricating said integrated...

...script to create said description language model of said integrated circuit design.

47 A method of generating the design of an integrated circuit using a **hardware description** language,
comprising the acts of:
selecting a process technology;
editing a first file specific to the design, said act of editing comprising selecting at least one user
configurable parameter selected from the group comprising;
(i) **processor** instructions;
(ii) cache **configuration**;

32/TI,PN,PY,PD,K/1 (Item 1 from file: 349)

METHOD AND SYSTEM FOR ADDING PHYSICAL DESIGN ANNOTATIONS TO A **HARDWARE DESCRIPTION LANGUAGE**
PROCEDE ET SYSTEME POUR AJOUTER DES ANNOTATIONS DE CONCEPTION PHYSIQUE A UN LANGAGE DE DESCRIPTION MATERIELLE
Patent and Priority Information (Country, Number, Date):
Patent: WO 9966431 A1 19991223
Publication Year: 1999
A system and method for providing a logic designer control over the physical design of a chip. A logical description of a module in a **hardware description language** is annotated with design directives. The design directives could relate, for example, to the synthesis, alignment and orientation, or physical dimension matching of the synthesized...

Claim
A logic design control method comprising:
creating a program in a **hardware description language** **describing** the logical
operation of a module;
annotating the program with design directives;
synthesizing the module into a set of one or more standard cells; and...

...relating to the synthesis of the module.

10 A system for chip design comprising:
a program describing the logical operation of a module in a **hardware**
description language;
annotations directing the physical design of the module included in the program; and
a **processor configured** to synthesize the module, and further configured to
generate a layout consistent with the annotations.

11 The system of claim 1 0 wherein:
the annotations a module in a **hardware description language**, the program being
annotated with design directives; synthesizing the module into a corresponding standard cell or set of standard
cells; and
generating a layout taking...

...of the synthesized module.

25 A system for chip design comprising:
means for receiving a program describing the logical operation of a module in a **hardware description language**, the program being annotated with design directives; means for synthesizing the module into a corresponding standard cell or set of
standard cells; and

32/TI,PN,PY,PD,K/2 (Item 2 from file: 349)

APPLICATION SPECIFIC PROCESSOR AND DESIGN METHOD FOR SAME
PROCESSEUR SPECIFIQUE A UNE APPLICATION ET SON PROCEDE DE CONCEPTION
Patent and Priority Information (Country, Number, Date):
Patent: WO 9531778 A1 19951123
Publication Year: 1995

or a class of said pre-designed elements, and including a user settable attribute defining a
selectable characteristic of said corresponding element; and
converting said **description** into a **hardware description language** suitable for use in the automated design of the physical structure of said integrated circuit.

39 The process of claim 38 wherein said integrated circuit comprises a clock and said selectable characteristic is the clocking speed of said bus.
40 The process of claim 38 further comprising synthesizing said **hardware description language** and fabricating said integrated circuit using a computer-aided design and manufacturing apparatus.
42 The process of claim 41 further comprising fabricating said processor using said **hardware description language** and loading said machine language instructions into one or more of the fabricated elements.

43 A method for designing a single chip integrated circuit application...

...bus, the method comprising the steps of. selecting a plurality of said hardware elements from said library in accordance with the desired functionality of said **processor;**
permanently **configuring** an attribute of said processor by defining said attribute
using one or more of said selected hardware elements; and
defining said processor using a **hardware description language**.

35/TI,PN,PY,PD,K/1 (Item 1 from file: 349)

A HIGH PERFORMANCE NETWORK INTERFACE
INTERFACE RESEAU HAUTE PERFORMANCE
Patent and Priority Information (Country, Number, Date):
Patent: WO 200052904 A1 20000908 (WO 0052904)
Publication Year: 2000

might be implemented using a variety of technologies. For example, the methods described herein may be implemented in software running on a programmable microprocessor, or **implemented** in **hardware** utilizing either a combination of microprocessors or other specially designed application specific integrated circuits, programmable logic devices, or various combinations thereof In particular, the methods... process the headers and deliver the data to the destination entity. Where the host computer includes multiple processors, a load distributor (which may also be **implemented** in **hardware** on the NIC) may select a processor to process the headers of the multiple packets through a protocol stack. In another embodiment of the invention...needed from a header is drawn from the correct portion of the header. Details concerning the form of a VLAN packet may be found in **specifications** for the IEEE 802. ...method by which a header parser may parse a header portion of a packet received at a network interface circuit from a network. In this **implementation**, the header parser is configured, or optimized, for parsing packets conforming to a set of pre-selected protocols (or protocol stacks). For packets meeting these...the invention virtually any form of data structure may be employed (e.g., database, table, queue, list, array), either monolithic or segmented, and may be **implemented** in **hardware** or software. The illustrated form of FDB I I 0 is merely one manner of maintaining usefal information concerning communication flows through NIC I 00...zero to sixty-three) to which the packet is to be submitted for processing. In this embodiment of the invention, load distributor II 2 is **implemented** in **hardware**, thus allowing rapid execution of the hashing and modulus functions. In an alternative embodiment of the invention, virtually any number of processors may be accommodated...

...to process a new packet is alerted or woken. In an embodiment of the invention operating on a SolariSTM workstation, individual processes executed by the **processor** are **configured** as "threads." A thread is a process running in a normal mode (e.g., not at an interrupt level) so as to have minimal impact...in control queue I 1 8 rather than packet queue I 1 6. In the illustrated embodiment of the invention packet queue II 6 is **implemented** in **hardware** (e.g., as random access memory). In this embodiment, checksum value 804 is sixteen bits in size and may be stored by checksum generator 114...tables may include a reference to, or some other means of identifying, a buffer. In FIG. I O@ DMA engine 120 is partially or ftilly **implemented** in **hardware** . Empty buffers into which packets may be stored are identified via a free descriptor 5 ring that is maintained in host memory. As one skilled ...within the scope of the invention as contemplated. For example, these data structures may take the form of arrays, lists, databases, etc., and may be **implemented** in **hardware** or softw

40/TI,PN,PY,PD,K/1 (Item 1 from file: 348)

Dynamic multi-mode parallel processor array architecture computer system
Dynamischer, in einer Array-Architektur im Mehrfachmodus arbeitender Parallelprozessor
Systeme d'ordinateur dynamique a architecture parallele multimode en forme de reseau
PATENT (CC, No, Kind, Date): EP 544127 A2 930602 (Basic)
EP 544127 A3 940420
EP 544127 B1 990310

...CLAIMS the system to a desired mode of operation by switching the processing mode bit (PMB) of the selected processors, said operating means enabling groups of **processors** to be dynamically **configured** to operate as either MIMD, SIMD, or SISD groups of processors.

2. A computer system according to claim 1 wherein said change of modes is ...interconnection network is utilized to pass broadcast messages and to interconnect a variety of subsets of processors within a group of processors, such that the **processors** may **implement** on a selective basis both SIMD and MIMD operations, some processors running SIMD and others running MIMD.

26. A computer system according to claim 1...

...to switch rapidly to and from MIMD mode, and even into SISD mode on the controlling processor for inherently sequential computation allows a programmer or **compiler** to build a program for the computer system which uses the optimal kind of parallelism (SISD, SIMD, MIMD) whichever make the most sense for the...

40/TI,PN,PY,PD,K/2 (Item 2 from file: 349)

INSTRUCTION PROCESSOR SYSTEMS AND METHODS
SYSTEMES ET PROCEDES DE PROCESSEURS D'INSTRUCTIONS
Patent and Priority Information (Country, Number, Date):
Patent: WO 200241146 A2 20020523 (WO 0241146)
Publication Year: 2002

The present invention relates to the design-time and run-time environments of instruction **processors implemented** in re-programmable **hardware**. In one aspect the present invention provides a design system for generating configuration information and associated executable code base on a customisation specification, which includes application information including application source code and customisation information including desgn constraints, for **implementing** an instruction **processor** using re-progammable hardware,t he system comprising: a template generator; an analyser; a **compiler**; an instantiator; and a builder. In another aspect the present invention provides a management system for managing run-time re-**configuration** of an instruction **processor implemented** using re-programmable **hardware**, comprising: a configuration library; a code library; a loader; a loader controller; a run-time monitor; an optimisation determiner; and an optimisation instructor.

Claim
... information and associated executable code based on a customisation specification, which includes application information including application source code and customisation information including design constraints, for **implementing** an instruction **processor** using re-programmable hardware, the system comprising:
a template generator for generating a template for each processor style identified as a candidate for implementation;
an analyser for analysing instruction information for each template and determining instruction optimisations;
a **compiler** for compiling the application source code to include the instruction optimisations and generate executable code;
an instantiator for analysing architecture information for each template, determining...
...of data and instruction paths.

33 The system of any of claims 1 to 32, wherein, where a plurality of configurations of the re-programmable **hardware** are required to **implement** the instruction **processor**, the instantiator is **configured** to optimise ones of the configurations into groups and schedule implementation of the grouped configurations.
38 The system of any of claims 1 to 36, wherein the **compiler** is configured to compile the application source code and re-organise the compiled source code to incorporate optimisations to provide an optimised executable code.
39...

...and
associated executable code, and, where relevant, the decision condition information, are deployed in at least one management system which is for managing adaptation and **configuration** of instruction **processors implemented** using re-programmable **hardware**.

..the configuration information and associated executable code, and, where relevant, the decision condition information, in at least one management system which is for managing re-**configuration** of instruction **processors**
implemented using re-programmable **hardware**.

...the re-programmable hardware comprises at least one complex programmable logic device. 140. The system of any of claims 87 to 139, wherein the instruction **processor** is fully **implemented** using the re-programmable hardware. 141. A method of managing run-time re-**configuration** of an instruction **processor implemented** in re-programmable **hardware**, comprising the steps ofproviding a configuration library containing configuration information for a plurality of instruction **processor implementations**;
providing a code library for containing associated executable code for the
implementations;

40/TI,PN,PY,PD,K/3 (Item 3 from file: 349)

SYSTEM AND METHOD FOR EXECUTING HYBRIDIZED CODE ON A DYNAMICALLY CONFIGURABLE HARDWARE ENVIRONMENT
SYSTEME ET PROCEDE D'EXECUTION D'UN CODE HYBRIDE SUR UN ENVIRONNEMENT MATERIEL CONFIGURABLE DE MANIERE DYNAMIQUE
Patent and Priority Information (Country, Number, Date):
Patent: WO 200203592 A2 20020110 (WO 0203592)
Publication Year: 2002
which argue against the function being identified as kernel sections include numerous branches and overly complex control code.
In an altemate embodiment, the **compiler** exanunes the code of the functions to deternune the size of arrays traversed and the number of variables that are live during the execution of...

...or function. Code that has less total memory used than that in the hardware accelerators and associated memories are classified as kernel code sections. The **compiler** may use weIl-understood optirnization methods such as constant propagation, ...overhead code. In altemate embodunents, the separation of step 652 may be performed manually by a programmer.
In step 656, the Figure 6 process creates **hardware** designs for **implementing** the kernel code sections of step 654. These designs are the executable code derived from the source code of the kernel code sections. Additionally, the...

...step 656 which allows for several embodiments.
In one embodirnent, the source code of the kernel code section is compiled automatically by one of several **compilers** corresponding to the available hardware accelerators. In an alternate embodiment, a progranimer may manually realize the executable code from the source code of the kernel...

...and the hardware accelerators, and both versions are loaded into the resulting binary. In a fourth embodiment, a hardware accelerator is synthesized into a custom. **hardware** accelerator **description**.
In step 658 the hardware designs of step 656 are mapped to all available target hardware. The target hardware may be a processor, an MCPE...

Returning now to decision step 652, the portions of the source code which were previously deemed overhead are sent to a traditional **compiler** 670 for compilation of object code to be executed on a traditional processor.
16
Altemately, the user may hand code the source program, into the...of a particular kernel code section9s execution in a bin. The functional units, arguments DMA 806 and results DMA 808 transfer this data without additional **processor** intervention. **Configuration** network source 8 1 0 controls the configuration network. 'Ihe configuration

network effects the configuration of the MCPEs of the bins bin 0 7069 bin ...request from the qucue written by a main processor, such as main processor 1304 of Figure 13. Then in step 1706 the run time kernel **processor** retrieves the **configuration** information needed to support execution of the requested kernel code segment. In step 1708 this information use used to build a new entry in a...one accelerator by manipulating the at least one set of configuration information.'

40/TI,PN,PY,PD,K/4 (Item 4 from file: 349)

METHOD AND APPARATUS TO IMPROVE CONTEXT SWITCH TIMES IN A COMPUTING SYSTEM
PROCEDE ET APPAREIL D'AMELIORATION DES TEMPS DE CHANGEMENT DE CONTEXTE DANS UN SYSTEME INFORMATIQUE

Patent and Priority Information (Country, Number, Date):
Patent: WO 200182059 A2-A3 20011101 (WO 0182059)
Publication Year: 2001

...reducing the amount of time required to perform a context switch. Program instructions are used to explicitly specify registers that are to be abandoned. For **processor**'s **implementing** a dirty bit scheme, the processor resets the dirty bits for those registers that are to be abandoned. A subsequent context switch excludes the abandoned...

Claim
... more registers; and
a process being executed by said processor, said process having a process context comprising one or more of said registers;
wherein said **processor** is **configured** to remove one or more of said
registers from said. process context in response to a first instruction associated with said process.

2 The apparatus of claim 1, wherein said **processor** is **configured** to
remove one or more of said registers from said process context by resetting one or more dirty bits associated with said one or more...

...one or more operands specifying one or more registers to be removed from said process context.

4 The apparatus of claim 3, further comprising a **compiler**, said **compiler** configured to:
determine whether a source register of a second instruction contains data that is needed after said second instruction is executed; and
set said...

...1, wherein said first instruction is a NOP instruction.

6 The apparatus of claim 1, wherein said first instruction comprises an abandon register bit, said **processor configured** to remove a register from said process context when said abandon register bit is set.

40/TI,PN,PY,PD,K/5 (Item 5 from file: 349)

PROGRAMMABLE SIGNAL PROCESSOR ARCHITECTURE
ARCHITECTURE DE PROCESSEUR DE SIGNAUX PROGRAMMABLE
Patent and Priority Information (Country, Number, Date):
Patent: WO 9118342 A1 19911128
Publication Year: 1991
..interface to the SPROC. SPROCs may be coupled via their input and output ports to provide a system. The SPROC architecture in conjunction with a **compiler** and user interface system permits a user to ''sketch and realize'' complex circuits in the SPROC. An access port (900) permits reading and writing to...

Claim
... said output port of said master and slave processor apparatuses, and parametric data for said data RAM of said master processor apparatus and said slave **processor** apparatuses, said microinstructions **configuration** information, and parametric data having been compiled into said programmed ROM, and wherein said master processor apparatus reads said microinstructions from said programmed ROM via...

...said microinstructions to respective program memory means of said master and slave processor apparatus for storage therein via respective program memory buses, and said master **processor** apparatus reads said **configuration** information from said programmed ROM via said master processor apparatus host port .and sends said configuration information appropriately to said data RAM, said data signal...said output means of said master and slave processor apparatuses, and parametric data for said data RAM of said master processor apparatus and said slave **processor** apparatuses, said microinstructions **configuration** information, and parametric data having been compiled into said programmed ROM, and wherein said master processor apparatus reads said microinstructions from said programmed ROM via...

07/31/2002 09/246,047

44/TI,PN,PY,PD,K/1 (Item 1 from file: 349)
DIALOG(R)File 349:(c) 2002 WIPO/Univentio. All rts. reserv.

VIDEO PROCESSING SYSTEM
SYSTEME DE TRAITEMENT VIDEO
Patent and Priority Information (Country, Number, Date):
Patent: WO 200199403 A2 20011227 (WO 0199403)
Publication Year: 2001
carncorder footage. The first few frames of home footage often contain garbage because the person has not quite set up the shot yet. In one **implementation**, a **better** choice for the representative frarne is 2a selected by analyzing the entire segment and selecting an image that best approximates the rest of the segment...

...digitized video to produce a streaming version (encoded version) that can be easily downloaded. or viewed. over a network (e.g., the Intemet). In one **implementation**, two parallel streaming **processors** 66a and. 66b are provided that produce streaming video output streams at two resolutions and bit rates. Streaming video processor 66a provides a streaming video output for supporting a 56k modem **configuration** while streaming video **processor** 66b provides a streaming video output for supporting a digital subscriber line (DSL) configuration. In one **implementation**, video output **processor** 66 outputs a Real.Video format file and. any accompanying SMIL files necessary for previewing the Real.Video fonnat file by the user. The output...

44/TI,PN,PY,PD,K/2 (Item 2 from file: 349)

METHOD AND APPARATUS FOR MONITORING A CACHE FOR GARBAGE COLLECTION
PROCEDE ET DISPOSITIF DE CONTROLE DE MEMOIRE CACHE EN VUE DE LA RECUPERATION D'ESPACE MEMOIRE
Patent and Priority Information (Country, Number, Date):
Patent: WO 200144947 A1 20010621 (WO 0144947)
Publication Year: 2001

Claim
... storage capability of a mass storage device and access performance approaching that of cache memory.
Figure 2 is a block diagram illustrating an example memory **configuration**. In Figure 2, **processor** 200 is coupled to a level one (LI) cache 203, which is in turn coupled to a level two (L2) cache 204. In this example...cache lines, while cache flushes of clean lines and cache line fills are handled separately by hardware. In other embodiments, the flush monitor may be **implemented** in **hardware** , or with a combination of hardware and software. The flush monitor is used to implement the write barrier for tracking inter-generational references between older...performed in software to flexibly implement the write barrier by appropriate setting of non-local bits.
However, in alternative embodiments, steps 502-507 may be **implemented** in **hardware** (e.g., where faster performance is desired) or in a combination of software and hardware. Cache line fills, regardless of whether the cache lines being...clearly decouples the cache flush and cache fill operations from the flush monitoring process. The cache flush and fill operations are handled immediately to provide **better** cache performance, whereas **implementation** of the write barrier by the flush monitor occurs at a more convenient time in the future. As long as a copy of the flushed...flushed to memory. The scanning operation may then be executed at a later time, based on the stored copy of the evicted cache line.
Cache **Hardware Implementation**
Figure 7 is a block diagram of a cache configuration in accordance with an embodiment of the invention. A direct-mapped cache configuration is shown...Otherwise, the access is performed directly through memory. Latency is high for direct memory access, but the cache remains unperturbed and no deadlock occurs.
For **processor** architectures **implementing** non-caching store instructions

44/TI,PN,PY,PD,K/3 (Item 3 from file: 349)

A HIGH PERFORMANCE NETWORK INTERFACE
INTERFACE RESEAU HAUTE PERFORMANCE
Patent and Priority Information (Country, Number, Date):
Patent: WO 200052904 A1 20000908 (WO 0052904)
Publication Year: 2000

. from a network and transferring it to a host computer system. In various embodiments of the invention, the high performance network interface is configured to **implement** one or more **enhanced** operations in order to efficiently handle a range of packet arrival rates without unduly burdening the host computer system. 5 One such operation is the...might be implemented using a variety of technologies. For example, the methods described herein may be implemented in software running on a programmable microprocessor, or **implemented** in **hardware** utilizing either a combination of microprocessors or other specially designed application specific integrated circuits, programmable logic devices, or various combinations thereof In particular, the methods ...process the headers and deliver the data to the destination entity. Where the host computer includes multiple processors, a load distributor (which may also be **implemented** in **hardware** on the NIC) may select a processor to process the headers of the multiple packets through a protocol stack. In another embodiment of the invention...method by which a header parser may parse a header portion of a packet received at a network interface circuit from a network. In this **implementation**, the header parser is configured, or optimized, for parsing packets conforming to a set of pre-selected protocols (or protocol stacks). For packets meeting these...the invention virtually any form of data structure may be employed (e.g., database, table, queue, list, array), either monolithic or segmented, and may be **implemented** in **hardware** or software. The illustrated form of FDB I I 0 is merely one manner of maintaining usefal information concerning communication flows through NIC I 00...zero to sixty-three) to which the packet is to be submitted for processing. In this embodiment of the invention, load distributor II 2 is **implemented** in **hardware**, thus allowing rapid execution of the hashing and modulus functions. In an alternative embodiment of the invention, virtually any number of processors may be accommodated...

...to process a new packet is alerted or woken. In an embodiment of the invention operating on a SolariSTM workstation, individual processes executed by the **processor** are **configured** as "threads." A thread is a process running in a normal mode (e.g., not at an interrupt level) so as to have minimal impact...in control queue I 1 8 rather than packet queue I 1 6. In the illustrated embodiment of the invention packet queue II 6 is **implemented** in **hardware** (e.g., as random access memory). In this embodiment, checksum value 804 is sixteen bits in size and may be stored by checksum generator 114...tables may include a reference to, or some other means of identifying, a buffer. In FIG. I O@ DMA engine 120 is partially or ftilly **implemented** in **hardware** . Empty buffers into which packets may be stored are identified via a free descriptor 5 ring that is maintained in host memory. As one skilled

...within the scope of the invention as contemplated. For example, these data structures may take the form of arrays, lists, databases, etc., and may be **implemented** in **hardware** or software. In the illustrated embodiment of the invention, header table 1006, MTU table 1008 and jumbo table IO 1 0 each contain only one...a CAM in this I 0 embodiment, all entries in the memory may be searched simultaneously or nearly simultaneously. In this embodiment, memory 2102 is **implemented** in **hardware**, with the entries logically arranged as a ring. In alternative embodiments, memory 2102 may be virtually any type of data structure (e.g., array, table, list, queue) **implemented** in **hardware** or software. In one particular alternative embodiment, memory 2102 is implemented as a 1 5 RAM, in which case the entries may be examined in...

44/TI,PN,PY,PD,K/4 (Item 4 from file: 349)

SYSTEM FOR DOWNLOADING AND REPORTING MEDICAL INFORMATION
SYSTEME POUR TELECHARGER ET TRANSMETTRE DES RENSEIGNEMENTS MEDICAUX
Patent and Priority Information (Country, Number, Date):
Patent: WO 9824358 A2 19980611
Publication Year: 1998

13 The system of claim 12 wherein:
said **processor** is further **configured** to establish a
connection over the network to facilitate real-time
communication between a health-care provider and a patient
concerning the results of'a medical measurement.

14 A system for **implementing** an **enhanced** interface
for a medical measurement,device, said system comprising:
a web appliance,, connected to a network and
including standard I/O ports and a display and for displaying interactive
function keys to
activate functions in the,tmedical device.

15 A system for **implementing** a **hardware** key for a
medical measurement device to gain access to a remote
database, said system comprising:
a web appliance, connected to a network and
including...

46/TI,PN,PY,PD,K/1 (Item 1 from file: 349)

INTEGRATED COMMERCE ENVIRONMENT (ICE) - A METHOD OF INTEGRATING OFFLINE AND ONLINE BUSINESS
ENVIRONNEMENT DE COMMERCE INTEGRE (ICE) UN PROCEDE D'INTEGRATION D'ENTREPRISE HORS LIGNE ET EN LIGNE
Patent and Priority Information (Country, Number, Date):
Patent: WO 200127838 A1 20010419 (WO 0127838)
Publication Year: 2001

to Fig. 14, a flow chart is presented to show some possible steps a merchant can follow in applying for PUMP service through a SAM1S **automated** signup **process**. Referring to Fig. 15, a flow chart is presented to show some possible steps a merchant can follow to program an existing standard credit-card...in signing up for PUMP service.

M. The access process is completed with the successful initiation of service operation.

N. An exemplary embodiment of an **Automated** Merchant Sign-up **Process** is shown in Fig. 14. Exemplary steps comprise one or more of the following:

a. The applicant fills out a form indicating all information technically ...cart computer system for use in said bricks-and-mortar business, wherein the shopping cart computer system comprises shopping cart hardware, comprising a shopping cart **processor** **and** **at least one input device that is coupled to said processor** and that is **configured** to collect at least one of profile information and identification information on at least one customer of an e-commerce business to build a profile...

50/TI,PN,PY,PD,K/1 (Item 1 from file: 349)

VIDEO SIGNAL WITH INTEGRAL DATA TRANSMISSION
SIGNAL VIDEO A TRANSMISSION DE DONNEES INTEGREE
Patent and Priority Information (Country, Number, Date):
Patent: WO 200223913 A2 20020321 (WO 0223913)
Publication Year: 2002

five error correction bits are appended to the data. The algorithm derived for the desired 29/24 code is p(x) X14 + x12 and is **implemented** in **hardware**, via a linear feedback shift register, in a conventional manner. Furthermore, since the data is a live uninterruptible stream, the linear feedback shift register, which...PC 42, however, does not calculate or append the CRC to the packet. Calculating and appending the CRC for the transmitter is preferably done in **hardware** as **described** below. Referring to FIG. 6, a circuit 130 for calculating and adding the CRC data to the end of a packet is shown. The circuit...the output of the mux 184 to the PC 162. Some of the information that is transmitted to the receiver and processed by the signal **processor** includes **configuration** information that is written from the host interface to the registers 188. As discussed herein, the registers 188 contain values for controlling 33 different aspects...line twenty and ending at line twenty-five, then the receiver will not detect the data sent by the transmitter. This occurs because the receiver **hardware**, **described** above, uses the configuration parameters to detect data in an expected portion of the I 0 video signal. The configuration parameters for the transmitter can...

50/TI,PN,PY,PD,K/2 (Item 2 from file: 349)

SYSTEM AND METHOD FOR EXECUTING HYBRIDIZED CODE ON A DYNAMICALLY CONFIGURABLE HARDWARE ENVIRONMENT
SYSTEME ET PROCEDE D'EXECUTION D'UN CODE HYBRIDE SUR UN ENVIRONNEMENT MATERIEL CONFIGURABLE DE MANIERE DYNAMIQUE
Patent and Priority Information (Country, Number, Date):
Patent: WO 200203592 A2 20020110 (WO 0203592)
Publication Year: 2002

. overhead code. In altemate embodunents, the separation of step 652 may be performed manually by a programmer.
In step 656, the Figure 6 process creates **hardware** designs for **implementing** the kernel code sections of step 654. These designs are the
executable code derived from the source code of the kernel code sections. Additionally, the...

...and the hardware
accelerators, and both versions are loaded into the resulting binary. In a fourth embodiment, a hardware accelerator is synthesized into a custom. **hardware** accelerator **description**.
In step 658 the hardware designs of step 656 are mapped to all available target hardware. The target hardware may be a processor, an MCPE...of a particular kernel code section9s execution in a bin. The functional units, arguments DMA 806 and results DMA 808 transfer this data without additional **processor** intervention. **Configuration** network source 8 1 0 controls the configuration network. 'Ihe configuration network effects the configuration of the MCPEs of the bins bin 0 7069 bin ...request from the qucue written by a main processor, such as main processor 1304 of Figure 13. Then in step 1706 the run time kernel **processor** retrieves the **configuration** information needed to support execution of the requested kernel code segment. In step 1708 this information use used to build a new entry in a...one accelerator by manipulating the at least one set of configuration information.'

50/TI,PN,PY,PD,K/3 (Item 3 from file: 349)

SCALEABLE ARCHITECTURE FOR MULTIPLE-PORT, SYSTEM-ON-CHIP ADSL COMMUNICATIONS SYSTEMS
ARCHITECTURE RECONFIGURABLE POUR SYSTEMES DE COMMUNICATION MULTIPORT A BOUCLE ADSL SUR UN MICROCIRCUIT
Patent and Priority Information (Country, Number, Date):
Patent: WO 200165774 A1 20010907 (WO 0165774)
Publication Year: 2001

Fulltext Availability:
Claims

English Abstract
A multi-port communications system (100) is **described**, which includes **hardware** based subsystems for performing both physical medium dependent operations and transport convergence operations on a data transmission. A software based subsystem performs other operations as...

Claim
... transport convergence operations (M) for said data transmission; and
wherein said first set of P@M operations and said first set of TC operations are
implemented entirely with a **hardware** based logic circuit;
performing a second set of PNM and/or a second set of TC operations for said
data transmission;
wherein said second set...

...said first set of transport convergence operations (TC).

42 The method of claim 41 3wherein said hardware based logic circuit and software based general purpose **processor** are **configurable** so that any of the following operating modes are supportable by the multi-port communications system: (1) one-port ADSL-Transceiver-Unit-Remote (ATU-R...

50/TI,PN,PY,PD,K/6 (Item 6 from file: 349)

SYSTEM FOR DIVIDING PROCESSING TASKS INTO SIGNAL PROCESSOR AND DECISION-MAKING MICROPROCESSOR INTERFACING
SYSTEME DE SEPARATION DES TACHES DE TRAITEMENT EN TACHES POUR INTERFACAGE AVEC UN PROCESSEUR DE SIGNAUX ET UN MICROPROCESSEUR DE PRISE DE DECISION

Patent and Priority Information (Country, Number, Date):
Patent: WO 9308524 A1 19930429
Publication Year: 1993

an environment which accounts for and provides connection and interfaces with the host microprocessor. It is another object of the invention to provide a programmable, **configurable**, real time signal **processor** which is particularly suited to the requirements of signal processing and which conducts deterministic real time signal processing and interfaces with a microprocessor which conducts...SPROC can be individually set or cleared via the parallel port, as described below. GP[3:01 are four general purpose pins that are individually **configurable** as either inputs or outputs. During reset, when RESET is LOW, all GPIO signals are set up as inputs. In addition to being subject to internal program control, the configuration of each GP pin, and the value of each GPIO signal **configured** as an output, are also individually controllable via the parallel port.
The SPROCboard evaluation board is a printed circuit board with one SPROC chip, digital-to-analog and analog-to-digital converters, and various... on the signal flow block diagram. In fact, the user could view all of the signals that are internal to each biquad. In the current **hardware implementation** of' the software-directed probe, values accessed using the probe command are made available as output from an on-chip, 8-bit, digital-to-analog...blocks. Code for signal processing functions is written at the primitive level. These primitive blocks comprise the SPROCcells function library, They are optimized for the **hardware** and efficiently **implemented** to extract maximum performance from the SPROC chip. Other primitive blocks include the glue blocks or phantoms required to provide. control and synchronization functions for...

53/TI,PN,PY,PD,K/1 (Item 1 from file: 348)

Method and apparatus for generating a hardware configuration display
Verfahren und Gerat zur Erzeugung der Anzeige einer Hardware-Konfiguration
Methode et appareil pour generer un affichage d'une configuration de materiel
PATENT (CC, No, Kind, Date): EP 675453 A1 951004 (Basic)
EP 675453 B1 011017

...ABSTRACT 50) and the processor (P, 40) and the system (5) having a display device (95) for displaying the configuration of the system (5). Within the **processor** (P, 40), a **configuration** display memory space (90) is created for constructing a display of the logical connections between the processor (P, 40) and the at least one device...

...CLAIMS devices (50, D) attached to said central processor, said apparatus comprising:
means for generating a configuration table (60, 70) in said memory of said central **processor**, said **configuration** table containing configuration data **describing** a current **hardware configuration** of said central **processor**, said **configuration** data defining logical connections between said one or more peripheral (50, D) and said central processor (40, P);
means for creating a configuration display memory...

...display of said logical connections between said one or more peripheral devices (50, D) and said central processor (40, P) by using said configuration data **describing** said current **hardware configuration** of said central **processor**; and
means (95) for outputting the contents of said configuration display memory space (90) to an output device to display the configuration of said system...

..displayable information;
creating configuration data relating to logical connections between said peripheral devices from a configuration table stored in said memory (10) of said central **processor** (40, P), said **configuration** table containing configuration data **describing** a current **hardware configuration** of said central **processor**;

53/TI,PN,PY,PD,K/2 (Item 2 from file: 349)

ANONYMOUS TRANSACTION SYSTEM
SYSTEME DE TRANSACTION ANONYME
Patent and Priority Information (Country, Number, Date):
Patent: WO 200242982 A2 20020530 (WO 0242982)
Publication Year: 2002

.. the functional blocks and software modules or features of the flexible interface call be implemented by themselves, or in combination with other operations in either **hardware** or software. Having **described** and illustrated the principles of the invention in a preferred embodiment thereof, it should be apparent that the invention can be modified in arrangement and...offers for goods and services with better terms than offers that the client has already accepted.

29 An anonymous transaction system, comprising:
one or more **processors configured** to receive first party data and convert the
first party data into an anonymous profile;
51
at least one of the **processors configured** to receive criteria from a second party
for providing a transaction;

53/TI,PN,PY,PD,K/3 (Item 3 from file: 349)

METHOD AND APPARATUS FOR MANAGING THE CONFIGURATION AND FUNCTIONALITY OF A SEMICONDUCTOR DESIGN
METHODE ET APPAREIL PERMETTANT DE GERER LA CONFIGURATION ET LA FONCTIONNALITE D'UN MODELE DE SEMI-CONDUCTEUR
Patent and Priority Information (Country, Number, Date):
Patent: WO 200022553 A2 20000420 (WO 0022553)
Publication Year: 2000

English Abstract
A method of managing the configuration, design parameters, and functionality of an integrated circuit (IC) design using a **hardware description** language (HDL). Instructions can be added, subtracted, or generated by the designer interactively during the design process, and customized HDL descriptions of the IC design...

Claim
... to create a customized description language model; and synthesizing said design based on said description language model.

2 The method of Claim 1, wherein said **description** language comprises a **hardware description** language (HDL).

53/TI,PN,PY,PD,K/5 (Item 5 from file: 349)

DYNAMICALLY RECONFIGURABLE HARDWARE SYSTEM FOR REAL-TIME CONTROL OF PROCESSES
SYSTEME MATERIEL RECONFIGURABLE DYNAMIQUEMENT QUI PERMET DE COMMANDER DES PROCESSUS EN TEMPS REEL
Patent and Priority Information (Country, Number, Date):
Patent: WO 9749042 A1 19971224
Publication Year: 1997

Fulltext Availability:
Claims

English Abstract
...13). The reconfigurable logic module (13) in communication with the configuration memory (19), establishes the hardware configuration in the module (13) on receipt of a **configuration** signal. The **processor** (17) is in communication over a data bus (27) with the reconfigurable logic module (13), the configuration memory (19), and sends the configuration signal to...

Claim
... and
(d) a communication port coupled to the reconfigurable logic module independently of signals on the data bus for controlling the external device;

2 A **hardware** system as **described** in claim 1. wherein the configuration memorylis
dual-ported random access memory.

53/TI,PN,PY,PD,K/6 (Item 6 from file: 349)

APPLICATION SPECIFIC PROCESSOR AND DESIGN METHOD FOR SAME
PROCESSEUR SPECIFIQUE A UNE APPLICATION ET SON PROCEDE DE CONCEPTION
Patent and Priority Information (Country, Number, Date):
Patent: WO 9531778 A1 19951123
Publication Year: 1995
or a class of said pre-designed elements, and including a user settable attribute defining a selectable characteristic of said corresponding element; and converting said **description** into a **hardware description** language suitable for use in the automated design of the physical structure of said integrated circuit.

39 The process of claim 38 wherein said integrated circuit comprises a clock and said selectable characteristic is the clocking speed of said bus.

40 The process of claim 38 further comprising synthesizing said **hardware description** language and fabricating said integrated circuit using a computer-aided design and manufacturing apparatus.

43 A method for designing a single chip integrated circuit...

...bus, the method comprising the steps of. selecting a plurality of said hardware elements from said library in accordance with the desired functionality of said **processor;**

47 The method of claim 43 further comprising synthesizing said **hardware description** language defining said processor using a computer-aided design tool.

07/31/2002 09/246,047

56/TI,PN,PY,PD,K/1 (Item 1 from file: 349)
DIALOG(R)File 349:(c) 2002 WIPO/Univentio. All rts. reserv.

INSTRUCTION PROCESSOR SYSTEMS AND METHODS
SYSTEMES ET PROCEDES DE PROCESSEURS D'INSTRUCTIONS
Patent and Priority Information (Country, Number, Date):
Patent: WO 200241146 A2 20020523 (WO 0241146)
Publication Year: 2002

English Abstract
The present invention relates to the design-time and run-time environments of instruction processors **implemented** in re-programmable **hardware**. In one aspect the present invention provides a design system for generating configuration information and associated executable code base on a customisation specification, which includes...

...generator; an analyser; a compiler; an instantiator; and a builder. In another aspect the present invention provides a management system for managing run-time re-**configuration** of an instruction **processor implemented** using re-programmable **hardware**, comprising: a configuration library; a code library; a loader; a loader controller; a run-time monitor; an optimisation determiner; and an optimisation instructor.
The AES **specification** [17] suggests that the AES algorithm could be accelerated by unrolling several of the AES functions into look-up-tables. Speeds of up to 7...

Claim
... of data and instruction paths.

33 The system of any of claims 1 to 32, wherein, where a plurality of configurations of the re-programmable **hardware** are required to **implement** the instruction **processor**, the instantiator is **configured** to optimise ones of the configurations into groups and schedule implementation of the grouped configurations.

34 The system of any of claims I to 33...and associated executable code, and, where relevant, the decision condition information, are deployed in at least one management system which is for managing adaptation and **configuration** of instruction **processors implemented** using re-programmable **hardware**.

...the configuration information and associated executable code, and, where relevant, the decision condition information, in at least one management system which is for managing re-**configuration** of instruction **processors** **implemented** using re-programmable **hardware**.

56/TI,PN,PY,PD,K/2 (Item 2 from file: 349)

A HIGH PERFORMANCE NETWORK INTERFACE
INTERFACE RESEAU HAUTE PERFORMANCE
Patent and Priority Information (Country, Number, Date):
Patent: WO 200052904 A1 20000908 (WO 0052904)
Publication Year: 2000

... might be implemented using a variety of technologies. For example, the methods described herein may be implemented in software running on a programmable microprocessor, or **implemented** in **hardware** utilizing either a combination of microprocessors or other specially designed application specific integrated circuits, programmable logic devices, or various combinations thereof In particular, the methods... process the headers and deliver the data to the destination entity. Where the host computer includes multiple processors, a load distributor (which may also be **implemented** in **hardware** on the NIC) may select a processor to process the headers of the multiple packets through a protocol stack. In another embodiment of the invention...needed from a header is drawn from the correct portion of the header. Details concerning the form of a VLAN packet may be found in **specifications** for the IEEE 802. ...the invention virtually any form of data structure may be employed (e.g., database, table, queue, list, array), either monolithic or segmented, and may be **implemented** in **hardware** or software. The illustrated form of FDB I I 0 is merely one manner of maintaining usefal information concerning communication flows through NIC I 00...zero to sixty-three) to which the packet is to be submitted for processing. In this embodiment of the invention, load distributor II 2 is **implemented** in **hardware**, thus allowing rapid execution of the hashing and modulus functions. In an alternative embodiment of the invention, virtually any number of processors may be accommodated...

59/TI,PN,PY,PD,K/1 (Item 1 from file: 348)

High performance frame time monitoring system and method for a fiber optic switch for a fiber optic network
Hochleistungs-Rahmenzeituberwachungssystem und Verfahren fur eine Lichtwellenleitervermittlung fur ein Lichtwellenleiternetz
Systeme et procede a haute performance de surveillance du temps de trame pour un commutateur a fibres optiques pour reseau a fibres optiques
PATENT (CC, No, Kind, Date): EP 724344 A2 960731 (Basic)
EP 724344 A3 980819

...ABSTRACT signals (146, 147) after the elapse of the periods, which can vary for optimization reasons depending upon frame class and type. Thus, in the foregoing **configuration**, a **processor** (100a) is utilized as a timing incrementer and logical decisions are allocated to the logic network (100b), resulting in an optimum balance between hardware and...

...CLAIMS for a fiber optic network, the network switch (30) for transferring the data frames (11) from source ports (33) to destination ports (33), comprising:

a **processor** (100a) **configured** to **implement** a timer (164) relative to an incoming frame (11) upon notification of receipt of said frame (11) by a source port, said **processor** (100a) **configured** to output sequential timer states (144); and

4. The system (100) of claim 1, wherein said logic network (100b) is **implemented** in **hardware** with an integrated circuit (100b) and wherein said timer (164) is implemented in software within said processor (100a).
5. A frame time monitoring system (100...

...frame time monitoring mechanism (100) configured to track time associated with said new data frame (121), said frame time monitoring mechanism (100) having:
(1) a **processor** (100a) **configured** to **implement** a timer (164) relative to said new data frame (11), said timer (164) being initiated by said sentry (104) when said new data frame (11) is determined to exist within said memory means (84), said **processor** (100a) **configured** to output timer states (144), said timer (164) being cleared via a clear signal (142) from said arbitrator (123) when said frame transfer request

59/TI,PN,PY,PD,K/2 (Item 1 from file: 349)

METHOD AND APPARATUS FOR MONITORING A CACHE FOR GARBAGE COLLECTION
PROCEDE ET DISPOSITIF DE CONTROLE DE MEMOIRE CACHE EN VUE DE LA RECUPERATION D'ESPACE MEMOIRE
Patent and Priority Information (Country, Number, Date):
Patent: WO 200144947 A1 20010621 (WO 0144947)
Publication Year: 2001

. storage capability of a mass storage device and access performance approaching that of cache memory.
Figure 2 is a block diagram illustrating an example memory **configuration**. In Figure 2, **processor** 200 is coupled to a level one (LI) cache 203, which is in turn coupled to a level two (L2) cache 204. In this example...cache lines, while cache flushes of clean lines and cache line fills are handled separately by hardware. In other embodiments, the flush monitor may be **implemented** in **hardware** , or with a combination of hardware and software. The flush monitor is used to implement the write barrier for tracking inter-generational references between older...performed in software to flexibly implement the write barrier by appropriate setting of non-local bits.
However, in alternative embodiments, steps 502-507 may be **implemented** in **hardware** (e.g., where faster performance is desired) or in a combination of software and hardware. Cache line fills, regardless of whether the cache lines being...flushed to memory. The scanning operation may then be executed at a later time, based on the stored copy of the evicted cache line.
Cache **Hardware Implementation**
Figure 7 is a block diagram of a cache configuration in accordance with an embodiment of the invention. A direct-mapped cache configuration is shown...Otherwise, the access is performed directly through memory. Latency is high for direct memory access, but the cache remains unperturbed and no deadlock occurs.
2 The system of claim 1, wherein said cache line fill is **implemented** in **hardware**.

07/31/2002 09/246,047 [redacted]

Set	Items	Description
S1	3243	CONFIGUR??????(3N)PROCESSOR? ?
S2	3243	CONFIGUR??????(3N)PROCESSOR??????
S3	1380	MC=S02-B08G
S4	24605	IC=G01C-021
S5	28062	S1:S4
S6	26200	(DESCRIPT?????? OR DESCRIB??????)(3N)HARDWARE
S7	8869	CC=B1265A Digital circuit design, modelling & testing
S8	80	MC=T01-G06C
S9	25427	HARDWARE(3N)IMPLEMENT??????
S10	9670	IMPLEMENT??????(3N)PROCESSOR? ?
S11	67223	S6:S10
S13	109	SYNTHES??????(3N)SCRIPT??????
S14	48	(PLACE OR ROUTE)(3N)SCRIPT??????
S15	14881	TEST??????(3N)BENCH? ?
S16	1641	CONFIGUR??????(3N)SPECIFICAT??????
S17	14676	SOFTWARE(3N)DEVELOP??????(3N)TOOL? ?
S18	120070	SOFTWARE(3N)DEVELOP??????
S19	64620	SOFTWARE(3N)TOOL? ?
S20	168114	S17:S19
S21	96268	COMPILER? ? OR ASSEMBLER? ? OR LINKER? ? OR DISASSEMBLER? ? OR DEBUGGER? ?
S22	196	INSTRUCTION(3N)SET(3N)SIMULATOR? ?
S23	34942	DIAGNOSTIC? ?(3N)TEST? ?
S24	19571	TEST??????(3N)TOOL? ?
S25	18918	(ENHANC? OR BETTER OR OPTIMUM OR OPTIMAL OR OPTIMIS?????? - OR OPTIMIZ??????)(3N)IMPLEMENT??????
S26	69429	AUTOMAT??????(3N)PROCESS? ?
S27	21453	(FAST?????? OR EXPEDIT??????)(3N)DEVELOP??????
S28	201	S5 AND S11
S29	3	S28 AND S12
S30	37187	HDL
S31	3	S28 AND S30
S32	0	S31 NOT S29
S33	198	S28 NOT S29
S34	6	S33 AND S20
S35	4	RD (unique items)
S36	194	S33 NOT S35
S37	192	S33 NOT S34
S38	15	S37 AND S21
S39	0	S38 AND S22
S40	0	S38 AND S23
S41	0	S38 AND S24
S42	0	S38 AND S25
S43	0	S38 AND (S25 OR S26 OR S27)
S44	13	RD S38 (unique items)
S45	0	IDPAT (sorted in duplicate/non-duplicate order)
S46	177	S37 NOT S38
S47	0	S46 AND S22
S48	0	S46 AND S23
S49	0	S46 AND S24
S50	171	S46 AND S1

STIC-EIC 2800 CP4-9C18 Irina Speckhard 308-6559

S51	34	S46 AND S6
S52	7	S51 AND S9
S53	3	RD (unique items)
S54	31	S51 NOT S53
S55	0	S54 AND S10
S56	0	S54 AND S17
S57	24	RD S54 (unique items)
S58	3	IDPAT (sorted in duplicate/non-duplicate order)
S59	3	IDPAT (primary/non-duplicate records only)
S60	47	S1 AND S6
S61	8	S60 AND S9
S62	4	RD (unique items)
S63	43	S60 NOT S62
S64	0	S63 AND S25
S65	0	S63 AND S26

07/31/2002 [redacted]

31jul02 12:45:12 User267149 Session D249.1

SYSTEM:OS - DIALOG OneSearch
File 2:INSPEC 1969-2002/Jul W4
(c) 2002 Institution of Electrical Engineers
*File 2: Alert feature enhanced for multiple files, duplicates removal, customized scheduling. See HELP ALERT.
File 6:NTIS 1964-2002/Aug W2
(c) 2002 NTIS, Intl Cpyrght All Rights Res
*File 6: Alert feature enhanced for multiple files, duplicates removal, customized scheduling. See HELP ALERT.
File 8:Ei Compendex(R) 1970-2002/Jul W4
(c) 2002 Engineering Info. Inc.
*File 8: Alert feature enhanced for multiple files, duplicates removal, customized scheduling. See HELP ALERT.
File 34:SciSearch(R) Cited Ref Sci 1990-2002/Jul W4
(c) 2002 Inst for Sci Info
*File 34: Alert feature enhanced for multiple files, duplicates removal, customized scheduling. See HELP ALERT.
File 434:SciSearch(R) Cited Ref Sci 1974-1989/Dec
(c) 1998 Inst for Sci Info
File 35:Dissertation Abs Online 1861-2002/Jun
(c) 2002 ProQuest Info&Learning
File 65:Inside Conferences 1993-2002/Jul W4
(c) 2002 BLDSC all rts. reserv.
File 77:Conference Papers Index 1973-2002/Jul
(c) 2002 Cambridge Sci Abs
File 94:JICST-EPlus 1985-2002/Jun W1
(c)2002 Japan Science and Tech Corp(JST)
*File 94: There is no data missing. UDs have been adjusted to reflect the current months data. See Help News94 for details.
File 99:Wilson Appl. Sci & Tech Abs 1983-2002/Jun
(c) 2002 The HW Wilson Co.
File 108:Aerospace Database 1962-2002/Jul
(c) 2002 AIAA
File 144:Pascal 1973-2002/Jul W4
(c) 2002 INIST/CNRS
File 238:Abs. in New Tech & Eng. 1981-2002/Jul
(c) 2002 Reed-Elsevier (UK) Ltd.
File 305:Analytical Abstracts 1980-2002/Jul W3
(c) 2002 Royal Soc Chemistry
*File 305: Alert feature enhanced for multiple files, duplicate removal, customized scheduling. See HELP ALERT.
File 315:ChemEng & Biotec Abs 1970-2002/Jan
(c) 2002 DECHEMA
File 350:Derwent WPIX 1963-2002/UD,UM &UP=200248
(c) 2002 Thomson Derwent
*File 350: Alerts can now have images sent vial all delivery methods. See HELP ALERT and HELP PRINT for more info.
File 347:JAPIO Oct 1976-2002/Mar(Updated 020702)
(c) 2002 JPO & JAPIO
*File 347: JAPIO data problems with year 2000 records are now fixed. Alerts have been run. See HELP NEWS 347 for details.
File 344:Chinese Patents Abs JuL 1985-2002/JuL

STIC-EIC 2800 CP4-9C18 Irina Speckhard 308-6559

07/31/2002 [redacted]

07/31/2002 [redacted]

29/3,AB/1 (Item 1 from file: 2)
DIALOG(R)File 2:INSPEC

6764971 INSPEC Abstract Number: B2001-01-6250F-005
Title: **Configurable** communications **processors** used for **implementing** a UMTS communications base station for mobile telephones
Author(s): Kleinmann, B.
Journal: Elektronik vol.49, no.22 p.64-70
Publisher: WEKA-Fachzeitschriften,
Publication Date: 31 Oct. 2000 Country of Publication: Germany
CODEN: EKRKAR ISSN: 0013-5658
SICI: 0013-5658(20001031)49:22L.64:CCPU;1-T
Material Identity Number: E071-2000-023
Language: German
Abstract: Illustrates the use of the reconfigurable communications processor type CS2112 from Chameleon Systems, with applications in wireless application protocol (WAP) systems for e-mail and web browsing. Shows how modulation and demodulation are achieved to connect the aerial RF front end elements with a vocoder through the **configurable** digital signal **processor** . Compares advantages of ASICs, FPGAs, DSPs and ASSPs (Applications-Specific Standard Products). Anticipates that the number of wireless Internet users world-wide will increase from 2 million in 1999 to 93 million by the year 2004. These mobile users will require full Personal Computer performance from their mobile systems. A reconfigurable signal processor from Chameleon company is described in detail. This can be programmed inn High level Languages such as C or **HDL**. A UMTS data block is shown, which has 15 time slots. Concludes that chip rate processing and symbol rate processing for a 24-channel system can be combined in a single type CS2112 integrated circuit.

07/31/2002 [redacted]

29/3,AB/2 (Item 1 from file: 8)
DIALOG(R)File 8:Ei Compendex(R)

06091258
E.I. No: EIP02287015270
Title: Design space exploration for DSP applications using the ASIP development system PEAS-III
Author: Kobayashi, Shinsuke; Mita, Kentaro; Takeuchi, Yoshinori; Imai, Masaharu
Corporate Source: Dept. of Informatics and Math. Sci. Osaka University, Osaka, Japan
Conference Title: 2002 IEEE International Conference on Acoustic, Speech, and Signal Processing
Conference Location: Orlando, FL, United States Conference Date: 20020513-20020517
E.I. Conference No.: 59257
Source: ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings v 3 2002. p III/3168-III/3171 (IEEE cat n 02ch37334)
Publication Year: 2002
CODEN: IPRODJ ISSN: 0736-7791
Language: English
Abstract: This paper describes rapid design space exploration for DSP applications using the PEAS-III system, which is a **configurable processor** development environment for application domain specific embedded systems. First, a compiler generation method, which is one of the key technologies in the PEAS-III system, is proposed. The target compiler is generated from the same information as used for the synthesizable **HDL** generation of the target processor. Using the PEAS-III system, not only the processor **HDL** description but also its target compiler are generated. Therefore, execution time which is computed from execution cycles of applications and generated processor's frequency can be rapidly evaluated. Experimental results showed that the trade-offs between area and performance of processors for DCT and FIR filter applications were analyzed in 4.1 hours and the optimal processor was selected under the design constraint by using generated compilers and processors. 9 Refs.

29/3,AB/3 (Item 1 from file: 94)
DIALOG(R)File 94:JICST-EPlus

05082661 JICST ACCESSION NUMBER: 02A0247666 FILE SEGMENT: JICST-E
A Compiler Generation Method in The PEAS-III System and Its Evaluation.
KOBAYASHI SHINSUKE (1); MITA KENTARO (1); TAKEUCHI YOSHINORI (1); IMAI MASAHARU (1)
(1) Osaka Univ. Graduate School of Engineering Sci., JPN
Denshi Joho Tsushin Gakkai Gijutsu Kenkyu Hokoku(IEIC Technical Report (Institute of Electronics, Information and Communication Enginners), 2002, VOL.101,NO.579(CPSY2001 91-104), PAGE.101-108, FIG.7, TBL.2, REF.9
JOURNAL NUMBER: S0532BBG
UNIVERSAL DECIMAL CLASSIFICATION: 681.3.068 681.325/.326.009.16
LANGUAGE: Japanese COUNTRY OF PUBLICATION: Japan
DOCUMENT TYPE: Journal
ARTICLE TYPE: Original paper
MEDIA TYPE: Printed Publication
ABSTRACT: This paper describes rapid design space exploration for DSP applications using the PEAS-III system, which is a **configurable processor** development environment for application domain specific embedded systems. First, a compiler generation method, which is one of the key technologies in the PEAS-III system, is proposed. The target compiler is generated from the same information used for the synthesizable **HDL** generation of the target processor. Using the PEAS-III system, not only the processor **HDL** description but also its target compiler are generated. Therefore, execution time, which is computed from execution cycles of applications and generated processors frequency, can be rapidly evaluated. Experimental results showed that the trade-offs among area, performance and power of processors for DCT and FIR filter applications were analyzed in 12.1 hours and the optimal processor was selected under the design constraint by using generated compilers. (author abst.)

07/31/2002 [redacted]

35/3,AB/1 (Item 1 from file: 2)
DIALOG(R)File 2:INSPEC

7175600 INSPEC Abstract Number: B2002-03-6135C-095, C2002-03-5260D-054
Title: Parameterizable hardware architectures for automatic synthesis of motion estimation processors
Author(s): Roma, N.; Sousa, L.
Author Affiliation: INESC-ID, Instituto Superior Tecnico, Lisbon, Portugal
Conference Title: 2001 IEEE Workshop on Signal Processing Systems. SiPS 2001. Design and Implementation (Cat. No.01TH8578) p.428-39
Editor(s): Catthoor, F.; Moonen, M.
Publisher: IEEE, Piscataway, NJ, USA
Publication Date: 2001 Country of Publication: USA viii+439 pp.
ISBN: 0 7803 7145 3 Material Identity Number: XX-2001-02283
U.S. Copyright Clearance Center Code: 0-7803-7145-3/01/$10.00
Conference Title: 2001 IEEE Workshop on Signal Processing Systems. SiPS 2001. Design and Implementation
Conference Sponsor: IEEE Signal Process. Soc.; IEEE Circuits & Syst. Soc
Conference Date: 26-28 Sept. 2001 Conference Location: Antwerp, Belgium
Language: English
Abstract: A new class of fully parameterizable multiple array architectures for motion estimation (ME) in video sequences based on the full search block matching (FSBM) algorithm is proposed in this paper. This class is based on a new and efficient AB2 single array architecture with minimum latency, maximum throughput and full utilization of the hardware resources. It provides the ability to **configure** the target **processors** according to the setup parameters, the processing time and the circuit area specified limits. With this purpose, a **software** configuration **tool** has been implemented to determine the set of possible configurations which fulfill the requisites of the video coder, providing the ability to automatically generate the VHDL description of the selected configuration. The implementation of a single array **processor configuration** on a single-chip is presented. Experimental results evidence the ability to estimate motion vectors in real-time with this configuration.

07/31/2002 [redacted]

35/3,AB/2 (Item 2 from file: 2)
DIALOG(R)File 2:INSPEC

4981057 INSPEC Abstract Number: C9508-6115-014
Title: VIOOL for hardware/software codesign
Author(s): Stoel, C.; Karrfalt, J.
Author Affiliation: Alternative Syst. Concepts Inc., Windham, NH, USA
p.333-40
Editor(s): Melhart, B.; Rozenblit, J.
Publisher: IEEE, New York, NY, USA
Publication Date: 1995 Country of Publication: USA xi+435 pp.
ISBN: 0 7803 2531 1
U.S. Copyright Clearance Center Code: 0 7803 2531 1/95/$4.00
Conference Title: Proceedings of the 1995 International Symposium and Workshop on Systems Engineering of Computer Based Systems
Conference Sponsor: IEEE Comput. Soc. Tech. Committee on the Eng. Comput. Based Syst. (ECBS); Electr. & Comput. Eng. Univ. Arizona
Conference Date: 6-9 March 1995 Conference Location: Tucson, AZ, USA
Language: English
Abstract: While searching for a methodology to allow the use of C++ as a universal design language, or UDL, while incorporating available VHDL implementations, it became clear that an object oriented design environment was needed. To facilitate this complex transition an environment called VHDL Interfacing Object-Oriented Languages (VIOOL) has been treated. VIOOL is a tool suite that allows the system designer (user) to model and simulate a complete system (i.e., hardware and software) in C++. VIOOL enables users to make tradeoffs in the hardware or software of a system based on performance information. The input to VIOOL is a hardware **configuration** (usually a **processor** model) described in VHDL, and the software written that is supposed to run on that hardware in C++. VIOOL is targeted towards application specific signal processor (ASSP) and digital signal processor (DSP) developers. Preliminary results show the feasibility of this approach.
Subfile: C

07/31/2002 [redacted]

35/3,AB/3 (Item 3 from file: 2)
DIALOG(R)File 2:INSPEC

02111592 INSPEC Abstract Number: B83051597, C83034773
Title: Interface **software development** for a digital cartridge tape drive
Author(s): Sri-Jayantha, M.; Miller, G.E.
Author Affiliation: Dept. of Mech. & Aerospace Engng., Princeton Univ., Princeton, NJ, USA
Journal: IEEE Transactions on Instrumentation and Measurement vol.IM-32, no.2 p.337-42
Publication Date: June 1983 Country of Publication: USA
CODEN: IEIMAO ISSN: 0018-9456
U.S. Copyright Clearance Center Code: 0018-9456/83/0600-0337$01.00
Language: English
Abstract: A microcomputer-based controller for a digital cartridge tape drive is developed. Minimum interface hardware is used to achieve the required control by making the best use of the I/O ports available on a single board computer. The software-intensive approach gives increased reliability and flexibility to the control scheme while keeping the development time short. A simple two-**processor** multiprocessing **configuration** is **implemented** and tested for a real-time data-acquisition system where the digital cartridge tape serves as the mass storage medium. Test/reset semaphores are used to coordinate the operation of the two processors. An initial application to real-time acquisition of aircraft flight data is described.

07/31/2002 [redacted]

44/3,AB/1 (Item 1 from file: 2)
DIALOG(R)File 2:INSPEC

7136232 INSPEC Abstract Number: B2002-02-1265F-011, C2002-02-5215-003
Title: A new hardware/software partitioning algorithm for DSP processor cores with two types of register files
Author(s): Togawa, N.; Sakurai, T.; Yanagisawa, M.; Ohtsuki, T.
Author Affiliation: Dept. of Inf. & Media Sci., The Univ. of Kitakyushu, Japan
Journal: IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences vol.E84-A, no.11 p.2802-7
Publisher: Inst. Electron. Inf. & Commun. Eng,
Publication Date: Nov. 2001 Country of Publication: Japan
CODEN: IFESEX ISSN: 0916-8508
SICI: 0916-8508(200111)E84A:11L.2802:HSPA;1-2
Material Identity Number: P710-2001-012
Language: English
Abstract: Given a compiled assembly code and a timing constraint of execution time, the proposed algorithm generates a **processor** core **configuration** with a new assembly code running on the generated processor core. The proposed algorithm considers two register files and determines the number of registers in each of the register files. Moreover the algorithm considers two or more types of functional units for each arithmetic or logical operation and assigns functional units with small area to a processor core without causing performance penalty. A generated processor core will have small area compared with processor cores which have a single register file or those which consider only one type of functional unit for each operation. The experimental results demonstrate the effectiveness and efficiency of the proposed algorithm.
Subfile: B C

07/31/2002 [redacted]

44/3,AB/2 (Item 2 from file: 2)
DIALOG(R)File 2:INSPEC

6788724 INSPEC Abstract Number: B2001-01-1265F-056, C2001-01-5130-030
Title: CAD tools for developing modularly **configured** attached **processors**
Author(s): Garcia, H.; Guerena, T.; Gibson, G.A.; Cabrera, S.; Mondragon, O.
Author Affiliation: Dept. of Electr. & Comput. Eng., Texas Univ., El Paso, TX, USA
Conference Title: 42nd Midwest Symposium on Circuits and Systems (Cat. No.99CH36356) Part vol. 1 p.321-4 vol. 1
Editor(s): Ramirez-Angulo, J.
Publisher: IEEE, Piscataway, NJ, USA
Publication Date: 2000 Country of Publication: USA 2 vol. (xl+1150) pp.
ISBN: 0 7803 5491 5 Material Identity Number: XX-2000-01816
U.S. Copyright Clearance Center Code: 0 7803 5491 5/99/$10.00
Conference Title: 42nd Midwest Symposium on Circuits and Systems
Conference Date: 8-11 Aug. 1999 Conference Location: Las Cruces, NM, USA
Language: English
Abstract: A computer-aided design tool, the SIMARC package, is used for the fast prototyping of modularly **configured** attached **processor** (MCAP) architectures. This tool consists of a graphics-user-interface (GUI) architecture editor, a GUI **assembler** tool, and a GUI simulator tool. This document presents the standard set of connections and component types that form the structure of a MCAP architecture. A new architecture for the conjugate gradient algorithm is introduced.
Subfile: B C

44/3,AB/3 (Item 3 from file: 2)
DIALOG(R)File 2:INSPEC

6646363 INSPEC Abstract Number: B2000-08-1265F-067, C2000-08-5215-020
Title: Co-synthesis to a hybrid RISC/FPGA architecture
Author(s): Gokhale, M.B.; Stone, J.M.; Gomersall, E.
Author Affiliation: Sarnoff Corp., Princeton, NJ, USA
Journal: Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology vol.24, no.2-3 p.165-80
Publisher: Kluwer Academic Publishers,
Publication Date: March 2000 Country of Publication: Netherlands
CODEN: JVSPED ISSN: 0922-5773
SICI: 0922-5773(200003)24:2/3L.165:SHRF;1-Y
Material Identity Number: G259-2000-003
U.S. Copyright Clearance Center Code: 0922-5773/2000/$18.00
Language: English
Abstract: Hybrid architectures combining conventional **processors** with **configurable** logic resources enable efficient coordination of control with datapath computation. With integration of the two components on a single device, housekeeping tasks and, optionally, loop control and data-dependent branching, can be handled by the conventional processor, while regular datapath computation occurs on the configurable **hardware**. This paper **describes** a novel approach to programming such hybrid devices that gives the programmer control over mapping of data and computation between conventional **processor** and **configurable** logic. With a simple set of pragma and intrinsic function directives, the NAPA C language provides for manual control over perhaps the most important aspect of programming such hybrid devices. Alternatively, as experience is gained about tradeoffs between the two computational resource, mapping directives may eventually be graduated by an external tool. The paper further describes a research prototype **compiler** that targets the hybrid processor model, with a concrete implementation for the National Semiconductor NAPA1000 chip. The NAPA C **compiler** parses the mapping directives, performs semantic analysis, and co-synthesizes a conventional processor executable combined with a configuration bit stream for the configurable logic. Two major **compiler** phases, the synthesis of pipelined loops and the datapath synthesis, are described in detail.
Subfile: B C

44/3,AB/4 (Item 4 from file: 2)
DIALOG(R)File 2:INSPEC

6358317 INSPEC Abstract Number: B1999-10-1265F-049, C1999-10-5135-011
Title: A hardware/software partitioning algorithm for processor cores of digital signal processing
Author(s): Togawa, N.; Sakurai, T.; Yanagisawa, M.; Ohtsuki, T.
Author Affiliation: Dept. of Electr., Inf. & Commun. Eng., Waseda Univ., Tokyo, Japan
Conference Title: Proceedings of the ASP-DAC '99 Asia and South Pacific Design Automation Conference 1999 (Cat. No.99EX198) Part vol.1 p. 335-8 vol.1
Publisher: IEEE, Piscatatway, NJ, USA
Publication Date: 1999 Country of Publication: USA (xxvi+372+suppl.) pp.
ISBN: 0 7803 5012 X Material Identity Number: XX-1999-00738
U.S. Copyright Clearance Center Code: 0 7803 5012 X/99/$10.00
Conference Title: Proceedings of the ASP-DAC '99 Asian and South Pacific Design Automation Conference 1999
Conference Sponsor: IEEE Hong Kong Sect. Comput. Chapter & CAS/COM Chapter; Hong Kong Inst. Eng. (Electron. Div.); City Univ. Hong Kong; Chinese Univ. Sci. & Technol.; Univ. Hong Kong
Conference Date: 18-21 Jan. 1999 Conference Location: Wanchai, Hong Kong
Language: English
Abstract: A hardware/software cosynthesis system for processor cores of digital signal processing has been developed. This paper focuses on a hardware/software partitioning algorithm which is one of the key issues in the system. Given an input assembly code generated by the **compiler** in the system, the proposed hardware/software partitioning algorithm first determines the types and the numbers of required hardware units, such as multiple functional units, hardware loop units, and particular addressing units, for a processor core (initial resource allocation). Second, the hardware units determined at initial resource allocation are reduced one by one while the assembly code meets a given timing constraint (**configuration** of a **processor** core). The execution time of the assembly code becomes longer but the hardware costs for a processor core to execute it becomes smaller. Finally, it outputs an optimized assembly code and a **processor configuration**. Experimental results demonstrate that the system synthesizes processor cores effectively according to the features of an application program/data.
Subfile: B C

07/31/2002 [redacted]

44/3,AB/5 (Item 5 from file: 2)
DIALOG(R)File 2:INSPEC

6034973 INSPEC Abstract Number: C9811-6150C-015
Title: NAPA C: compiling for a hybrid RISC/FPGA architecture
Author(s): Gokhale, M.B.; Stone, J.M.
Conference Title: Proceedings. IEEE Symposium on FPGAs for Custom Computing Machines (Cat. No.98TB100251) p.126-35
Editor(s): Pocek, K.L.; Arnold, J.M.
Publisher: IEEE Comput. Soc, Los Alamitos, CA, USA
Publication Date: 1998 Country of Publication: USA x+344 pp.
ISBN: 0 8186 8900 5 Material Identity Number: XX98-02491
U.S. Copyright Clearance Center Code: 0 8186 8900 5/98/$10.00
Conference Title: Proceedings IEEE Symposium on FPGAs for Custom Computing Machines
Conference Sponsor: IEEE Comput. Soc. Tech. Committee on Comput. Architecture
Conference Date: 15-17 April 1998 Conference Location: Napa Valley, CA, USA
Language: English
Abstract: Hybrid architectures combining conventional **processors** with **configurable** logic resources enable efficient coordination of control with datapath computation. With integration of the two components on a single device, loop control and data-dependent branching can be handled by the conventional processor. While regular datapath computation occurs on the configurable **hardware**. This paper **describes** a novel pragma-based approach to programming such hybrid devices. The NAPA C language provides pragma directives so that the programmer (or an automatic partitioner) can specify where data is to reside and where computation is to occur with statement-level granularity. The NAPA C **compiler**, targeting National Semiconductor's NAPA1000 chip, performs semantic analysis of the pragma-annotated program and co-synthesizes a conventional program executable combined with a configuration bit stream for the adaptive logic. **Compiler** optimizations include synthesis of hardware pipelines from pipelineable loops.
Subfile: C

07/31/2002 [redacted]

44/3,AB/6 (Item 6 from file: 2)
DIALOG(R)File 2:INSPEC

5369079 INSPEC Abstract Number: A9620-2960-010, B9610-7430-071
Title: Enable++: a general-purpose L2 trigger processor
Author(s): Hogl, H.; Kugel, A.; Ludvig, J.; Manner, R.; Noffz, K.H.; Zoz, R.
Author Affiliation: Lehrstuhl fur Inf. V, Mannheim Univ., Germany
Conference Title: 1995 IEEE Nuclear Science Symposium and Medical Imaging Conference Record (Cat. No.95CH35898) Part vol.2 p.667-71 vol.2
Editor(s): Moonier, P.A.
Publisher: IEEE, New York, NY, USA
Publication Date: 1995 Country of Publication: USA 3 vol. li+1814 pp.
ISBN: 0 7803 3180 X Material Identity Number: XX96-01974
U.S. Copyright Clearance Center Code: 0 7803 3180 X/96/$5.00
Conference Title: 1995 IEEE Nuclear Science Symposium and Medical Imaging Conference Record
Conference Date: 21-28 Oct. 1995 Conference Location: San Francisco, CA, USA
Language: English
Abstract: Two years of experience with the two prototype FPGA processors Enable-1 and DecPeRLe-1 reveal that field programmable processors are the best choice for realizing a data-driven second level (L2) trigger for ATLAS. This paper presents Enable++, a modular and thus scalable 2nd generation FPGA processor that offers several substantial enhancements to the previous systems: In order to meet the varying demands of all ATLAS subdetectors Enable++ is structured into three different state-of-the-art modules for providing computing power, flexible and high-speed I/O communication and powerful intermodule communication with a raw bandwidth of 3.2 GByte/s by an active backplane. The computing core offers scalable computing power by virtue of a **configurable processor** topology, a 4*4 FPGA array and 12 MByte of distributed RAM. For building new applications the system provides a comfortable programming and debugging environment consisting of a **compiler** for the C-like **hardware description** language spC, a simulator and a source level **debugger** for hardware design. The most computing intensive tasks in L2 triggering are the feature extraction algorithms. From experience with Enable-1 we expect that Enable++ surpasses modern RISC processors by a factor of 100 to 1000.
Subfile: A B

07/31/2002 [redacted]

44/3,AB/7 (Item 7 from file: 2)
DIALOG(R)File 2:INSPEC

5162747 INSPEC Abstract Number: B9602-1265B-172, C9602-5400-001
Title: Enable++: a second generation FPGA processor
Author(s): Hogl, H.; Kugel, A.; Ludvig, J.; Manner, R.; Noffz, K.H.; Zoz, R.
Author Affiliation: Mannheim Univ., Germany
Conference Title: Proceedings IEEE Symposium on FPGAs for Custom Computing Machines (Cat. No.95TB8077) p.45-53
Editor(s): Athanas, P.; Pocek, K.L.
Publisher: IEEE Comput. Soc. Press, Los Alamitos, CA, USA
Publication Date: 1995 Country of Publication: USA viii+222 pp.
ISBN: 0 8186 7086 X Material Identity Number: XX95-03180
U.S. Copyright Clearance Center Code: 0 8186 7086 X/95/$04.00
Conference Title: Proceedings IEEE Symposium on FPGAs for Custom Computing Machines
Conference Sponsor: IEEE Comput. Soc. Tech. Committee on Comput. Architecture
Conference Date: 19-21 April 1995 Conference Location: Napa Valley, CA, USA
Language: English
Abstract: In the computing community field, programmable processors are going to fill the niche for special purpose computing devices. A typical example is ultra fast pattern recognition in experimental particle physics, a task for which we constructed two years ago (1993), Enable 1, an FPGA processor rather specialized for pattern recognition algorithms in mu s domain, but also provided with modest features for coping with more general applications. The paper presents the follow up model Enable++, a 2nd generation FPGA processor that offers several substantial enhancements over the previous system for a wider range of applications: Enable++ is structured into three different state of the art modules for providing computing power, flexible and high speed I/O communication and powerful intermodule communication with a raw bandwidth of 3.2 GByte/s by an active backplane. The technical realization of all three modules is guided by the maximum usage of field programmable logic. The actual demand of computing and I/O power can be satisfied by the number of modules plugged into the crate. Enhanced features of Enable++ comprise the **configurable processor** topology provided by programmable crossbar switches. In combination with the 4*4 FPGA array and 12 MByte distributed RAM, the Enable++ computing core offers a strongly increased and scalable computing power. For building new applications, the system offers a comfortable programming and debugging environment consisting of a **compiler** for the C like **hardware description** language spC, a simulator and a source level **debugger** for hardware design

44/3,AB/8 (Item 1 from file: 6)
DIALOG(R)File 6:NTIS

1523823 NTIS Accession Number: AD-A223 415/1
Ada **Compiler** Validation Summary Report: Certificate Number: 900121S1.10251 Computer Sciences Corporation MC Ada V1.2.Beta/Concurrent Computer Corporation Concurrent/Masscomp 5600 Host To Concurrent/Masscomp 5600 (Dual 68020 **Processor Configuration**) Target
(Final rept)
National Inst. of Standards and Technology, Gaithersburg, MD.
Corp. Source Codes: 092732000; 419591
23 Apr 90 50p
Languages: English
Journal Announcement: GRAI9021
Order this product from NTIS by: phone at 1-800-553-NTIS (U.S. customers); (703)605-6000 (other countries); fax at (703)321-8547; and email at orders@ntis.fedworld.gov. NTIS is located at 5285 Port Royal Road, Springfield, VA, 22161, USA.
NTIS Prices: PC A03/MF A01
This Validation Summary Report describes the extent to which a specific Ada **compiler** conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this **compiler** using the Ada **Compiler** Validation Capability. An Ada **compiler** must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard. Even though all validated Ada **compilers** conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between **compilers** result from the characteristics of particular operating systems, **hardware**, or **implementation** strategies. All the dependencies observed during the process of testing this **compiler** are given in this report. The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada **compiler** and evaluating the results. (kr)

07/31/2002 [redacted]

44/3,AB/9 (Item 2 from file: 6)
DIALOG(R)File 6:NTIS

1451118 NTIS Accession Number: AD-A208 766/6
Ada **Compiler** Validation Summary Report: Certificate Number: 881208W1.10010, BiiN, BiiN Ada, Version 2.00, BiiN 60 (8 **Processor Configuration**) Host/Target
Information Systems and Technology Center, Wright-Patterson AFB, OH. ADA Validation Facility.
Corp. Source Codes: 085490001; 416049
Report No.: AVF-VSR-213.0289
6 Dec 88 56p
Languages: English
Journal Announcement: GRAI8919
Order this product from NTIS by: phone at 1-800-553-NTIS (U.S. customers); (703)605-6000 (other countries); fax at (703)321-8547; and email at orders@ntis.fedworld.gov. NTIS is located at 5285 Port Royal Road, Springfield, VA, 22161, USA.
NTIS Prices: PC A04/MF A01
This Validation Summary Report describes the extent to which a specific Ada **compiler** conforms to the Ada Standard , ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this **compiler** using the Ada **Compiler** Validation Capability. An Ada **compiler** must be implemented according to the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard. Even though all validated Ada **compilers** conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between **compilers** results from the characteristics of particular operating systems, **hardware**, or **implementation** strategies. All the dependencies observed during the process of testing this **compiler** are given in this report. The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada **compiler** and evaluating the results. (kr)

07/31/2002 [redacted]

44/3,AB/10 (Item 1 from file: 8)
DIALOG(R)File 8:Ei Compendex(R)

04495854
E.I. No: EIP96093325879
Title: Enable plus plus : a general-purpose L2 trigger processor
Author: Hoegl, H.; Kugel, A.; Ludvig, J.; Maenner, R.; Noffz, K.H.; Zoz, R.
Corporate Source: Universitaet Mannheim, Mannheim, Ger
Conference Title: Proceedings of the 1995 IEEE Nuclear Science Symposium and Medical Imaging Conference. Part 2 (of 3)
Conference Location: San Francisco, CA, USA Conference Date: 19951021-19951028
E.I. Conference No.: 45267
Source: IEEE Nuclear Science Symposium & Medical Imaging Conference v 2 1995. IEEE, Piscataway, NJ, USA,95CH35898. p 667-671
Publication Year: 1995
CODEN: 85OQAD
Language: English
Abstract: Two years of experience with the two prototype FPGA processors Enable-1 and DecPeRLe-1 reveal that field programmable processors are the best choice for realizing a data-driven second level (L2) trigger for ATLAS. This paper presents Enable plus plus , a modular and thus scalable 2nd generation FPGA processor that offers several substantial enhancements to the previous systems: In order to meet the varying demands of all ATLAS subdetectors Enable plus plus is structured into three different state-of-the-art modules for providing computing power, flexible and high-speed I/O communication and powerful intermodule communication with a raw bandwidth of 3.2 GByte/s by an active backplane. The computing core offers scalable computing power by virtue of a **configurable processor** topology, a 4 multiplied by 4 FPGA array and 12 MByte of distributed RAM. For building new applications the system provides a comfortable programming and debugging environment consisting of a **compiler** for the C-like **hardware description** language spC, a simulator and a source level **debugger** for hardware design. The most computing intensive tasks in L2 triggering are the feature extraction algorithms. From experience with Enable-1 we expect that Enable plus plus surpasses modern RISC processors by a factor of 100 to 1000. (Author abstract) Refs.

07/31/2002 [redacted]

44/3,AB/11 (Item 2 from file: 8)
DIALOG(R)File 8:Ei Compendex(R)

04368408
E.I. No: EIP96033111814
Title: C plus plus **compiler** for FPGA custom execution units synthesis
Author: Iseli, Christian; Sanchez, Eduardo
Corporate Source: Ecole Polytechnique Federale, Lausanne, Switz
Conference Title: Proceedings of the 1995 IEEE Symposium on FPGAs for Custom Computing Machines
Conference Location: Napa Valley, CA, USA Conference Date: 19950419-19950421
E.I. Conference No.: 44424
Source: IEEE Symposium on FPGAs for Custom Computing Machines, Proceedings 1995. IEEE, Piscataway, NJ, USA,95CS8077. p 173-179
Publication Year: 1995
CODEN: 002320
Language: English
Abstract: If reconfigurable processors are to become widely used, we will need tools to help conventional programmer use them. In particular, a single high-level language should be used to program the whole application; both the part which will become the hardware configuration and the part which remains software. Spyder is a reconfigurable **processor** with **configurable** execution units. The C plus plus language has been chosen as the source language to program this processor. In this paper we present a **compiler** capable of synthesizing the hardware configuration of FPGA execution units from C plus plus source code. The same source code can be compiled by a standard C plus plus **compiler** for simulation purposes. First estimates show that this approach leads to very short synthesize time as compared to VHDL synthesizer for a similar quality of the generated hardware. (Author abstract) 11 Refs.

44/3,AB/12 (Item 1 from file: 35)
DIALOG(R)File 35:Dissertation Abs Online

01681742 AAD9914723
INSTRUCTION SCHEDULING AND FETCH MECHANISMS FOR CLUSTERED VLIW PROCESSORS (CLUSTERED MICROPROCESSORS)
Author: BANERJIA, SANJEEV
Degree: PH.D.
Year: 1998
Corporate Source/Institution: NORTH CAROLINA STATE UNIVERSITY (0155)
Source: VOLUME 59/12-B OF DISSERTATION ABSTRACTS INTERNATIONAL.
PAGE 6376. 121 PAGES

The design of high-performance microprocessors involves the use of innovative architectural and micro-architectural techniques and aggressive technology. As issue widths increase to enable greater degrees of instruction-level parallelism (ILP), the number of ports on the register file must also increase to allow all operations that are issued simultaneously to read and write their operands. However, it is difficult to construct a register file with a large number of read ports without stretching the cycle time of the processor. An alternative is to build processors that use multiple, disjoint register files instead of central, monolithic ones. Sets of register files and functional units (FU) can be grouped together as <italic>clusters</italic>, such that multiple clusters constitute the entire processor.

When an operation executing on a clustered processor requires (register) source operands that reside on different clusters, some form of inter-cluster communication must be used so that the operation can read its source operands. For a VLIW architecture, the **compiler** must manage the data movements between the clusters. As a clustered organization becomes more important in the design of present-day processors, especially VLIW processors, **compiler** technology must manage the intercluster data movements. This thesis explores three key issues germane to the design of clustered processors:

<italic>Cluster scheduling</italic>. This is the compile-time scheduling of inter-cluster data movements. The most prominent prior work in the area is the Bottom-up Greedy (BUG) algorithm from Ellis's PhD work and used in the **compiler** for the Multiflow TRACE family of commercial VLIW machines. A new algorithm is presented that performs the assignment of operations to clusters and schedules all operations and data movements within a single **compiler** phase. The algorithm is named Unified-Assign-and-Schedule or UAS. The algorithm is shown to be effective at producing high-performance code schedules with less code expansion than BUG.

<italic>Evaluating different clustered machine models</italic>. A wide variety of machine models can be used when **implementing** a clustered **processor**. However, certain **configurations** are certainly more appropriate than others. A range of machine models are simulated to gauge what models are best-suited as clustered processors.

<italic>Instruction fetch for clustered processors</italic>. Fetching instructions across a wide clustered machine is not as straightforward as on a non-clustered machine. However, there are issues common to both types

of machines. I-fetch for non-clustered VLIWs is presented to highlight the critical points. One important issue is how branch operations are executed on a clustered VLIW machine. The Prepare-To-Branch (PBR) branch architecture is reviewed and applied to the address the problem of a taken-branch penalty. Two hardware structures, the NextPC queue and the Parallel NextPC (P-NextPC) queue, are introduced as mechanisms that permit maximum **compiler** scheduling freedom for the PBR architecture. Also, b-caches (branch caches) and branch replication are described as methods for overcoming the latency for transmitting a PC across a clustered machine.

The thesis concludes by discussing ideas for future work in the three areas.

07/31/2002 [redacted]

44/3,AB/13 (Item 1 from file: 94)
DIALOG(R)File 94:JICST-EPlus

01267305 JICST ACCESSION NUMBER: 91A0200765 FILE SEGMENT: JICST-E
S-S300 RISC workstation.
KAWAMURA K (1); ASAINA T (1); KANAI H (1); SUZUKI K (1); OOKA T (1); KOMEDA S (1); TSUJIMOTO K (1)
(1) SUMITOMO ELECTRIC
Sumitomo Electr Tech Rev, 1991, NO.31, PAGE.92-96, FIG.5, TBL.1
JOURNAL NUMBER: S0886AAM ISSN NO: 0376-1207
UNIVERSAL DECIMAL CLASSIFICATION: 681.327.8
LANGUAGE: English COUNTRY OF PUBLICATION: Japan
DOCUMENT TYPE: Journal
ARTICLE TYPE: Original paper
MEDIA TYPE: Printed Publication
ABSTRACT: The subject work station which installed a RISC processor was introduced. This paper **describes** the following ;1) **hardware configurations** (CPU, floating point **processor**, cash memory, main memory interface, standard I/O, and bit map display),2) software (OS, **compiler**, networks and graphics/window),3) performance,4) applications. As a RISC processor, 21MIPS R3000 (MIPS Co.) was adopted. As an OS, the SEIUX/V.3.1 with a Berkeley edition UNIX environment was used.

07/31/2002 [redacted]

53/3,AB/1 (Item 1 from file: 2)
DIALOG(R)File 2:INSPEC

02894398 INSPEC Abstract Number: C87034906
Title: Multiprocessor Unix operating systems
Author(s): Bach, M.J.; Buroff, S.J.
Author Affiliation: AT&T Bell Labs., Murray Hill, NJ, USA
Book Title: UNIX system readings and applications. Vol.II. The UNIX system p.151-67
Publisher: Prentice-Hall, Englewood Cliffs, NJ, USA
Publication Date: 1987 Country of Publication: USA xii+324 pp.
ISBN: 0 13 939845 7
Language: English
Abstract: This paper describes the problems posed by running the Unix operating system on multiprocessors, as well as some solutions. The resulting systems function like their single-processor counterparts but yield 70 percent better throughput for two-**processor configurations**. Closely coupled multiprocessor Unix systems currently run on IBM and AT&T Technologies **hardware**, but the **implementation described** in this paper ports to other architectures as well, and the design is not limited to two-**processor configurations**.

07/31/2002 [redacted]

53/3,AB/2 (Item 2 from file: 2)
DIALOG(R)File 2:INSPEC

01362403 INSPEC Abstract Number: A79052797, C79019206
Title: Small laboratory computer networks
Author(s): Dessy, R.E.
Author Affiliation: Chem. Dept., Virginia Polytech. Inst., Blacksburg, VA, USA
Journal: Analytica Chimica Acta, Computer Techniques and Optimization vol.103, no.4 p.459-68
Publication Date: 15 Dec. 1979 Country of Publication: Netherlands
CODEN: CTOPD3 ISSN: 0378-4304
Conference Title: Computers and Optimization in Analytical Chemistry
Conference Date: 5-7 April 1978 Conference Location: Amsterdam, Netherlands
Language: English
Abstract: This permits program preparation and data manipulation/storage/plotting/printing to take place on the CPU most appropriate to the task. It also allows the application programs to run on satellite **processors** with minimum **configuration** and maximum data throughput capabilities. The **hardware implementation** is **described**, as well as a brief introduction to the software facilities available under RT-11/REMOTE and FORTH.
Subfile: A C

07/31/2002 [redacted]

53/3,AB/3 (Item 3 from file: 2)
DIALOG(R)File 2:INSPEC

01144360 INSPEC Abstract Number: B78007306
Title: GDM/GPS receiver **hardware implementation**
Author(s): Bjornsen, G.L.; Hutchinson, W.M.
Author Affiliation: Collins Avionics Div., Rockwell International, Cedar Rapids, IA, USA
Conference Title: Proceedings of the IEEE 1977 National Aerospace and Electronics Conference, NAECON '77 p.303-9
Publisher: IEEE, New York, NY, USA
Publication Date: 1977 Country of Publication: USA xxxi+1369 pp.
Conference Sponsor: IEEE; et al
Conference Date: 17-19 May 1977 Conference Location: Dayton, OH, USA
Language: English
Abstract: **Describes** the receiver **hardware implementation%%** **% for the AFAL GDM/GPS equipment. Included are descriptions of the RF receiver, frequency synthesizer, and signal channel processor. Specific items discussed include receiver bandwidth, wide-band AGC performance, pseudonoise (PN) mixers, code correlation, PN code generator, and digital voltage controlled oscillators. Various system issues, as they relate to the GDM/GPS equipment are also addressed. The GDM/GPS hardware has been partitioned such that it can be** configured **(under** processor **control) to represent various GPS equipment configurations for performance evaluation. A brief description of an internally generated test signal and its use for system calibration is also included.**
Subfile: B

59/3,AB/3 (Item 3 from file: 350)
DIALOG(R)File 350:Derwent WPIX

010437162
WPI Acc No: 1995-338479/199544
XRPX Acc No: N95-253894
Processor system with system configuration display - has memory space which constructs display of logical connections between processor and external device
Patent Assignee: INT BUSINESS MACHINES CORP (IBMC); IBM CORP (IBMC)
Inventor: GNIRSS M; REDDEMANN B; SCHUEPPEN W; SCHUPPEN W
Number of Countries: 005 Number of Patents: 005
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
EP 675453	A1	19951004	EP 94105160	A	19940331	199544	B
JP 7271708	A	19951020	JP 94319906	A	19941222	199551	
US 5627955	A	19970506	US 95412809	A	19950329	199724	
EP 675453	B1	20011017	EP 94105160	A	19940331	200169	
DE 69428681	E	20011122	DE 628681	A	19940331	200201	
			EP 94105160	A	19940331		

Priority Applications (No Type Date): EP 94105160 A 19940331
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
EP 675453	A1	E	21	G06F-017/50	
Designated States (Regional): DE FR GB					
JP 7271708	A		15	G06F-013/14	
US 5627955	A		20	G06F-015/00	
EP 675453	B1	E		G06F-017/50	
Designated States (Regional): DE FR GB					
DE 69428681	E			G06F-017/50	Based on patent EP 675453

Abstract (Basic): EP 675453 A

The processor system includes a processor with a configuration table which defines its logical connections to a device. A display shows the configuration. The processor also includes a configuration display memory space which constructs a display of the logical connections. An output device such as a printer shows the contents of the display memory space.

The system may also include several other processors and devices, and also control units. Connections between all of these are stored in various configuration tables. Pointers are used to indicate the lists of processors and devices in the tables.

ADVANTAGE - Allows production of complex graphical displays by user.

Dwg.5/20

Abstract (Equivalent): US 5627955 A

A method for displaying the hardware configuration of a data processing system on an output device, said data processing system comprising a plurality of entities having a defined topology, said entities including a central processor having an associated memory, said method comprising the steps of:

defining displayable information;

creating a configuration display memory space in said memory of said central processor;

reading configuration data relating to logical connections between said entities from a configuration table stored in said memory of said central processor, said configuration table containing configuration data describing a current hardware configuration of said central processor;

determining said topology of said entities;

placing in said configuration display memory space entries to represent said entities;

extracting connection data between said entities from the read configuration data;

creating in said configuration display memory space connections between said entities using the extracted connection data; and

outputting on said output device the configuration of said data processing system using said configuration display memory space.

Dwg.5/20b

62/3,AB/2 (Item 2 from file: 2)
DIALOG(R)File 2:INSPEC

01362403 INSPEC Abstract Number: A79052797, C79019206
Title: Small laboratory computer networks
Author(s): Dessy, R.E.
Author Affiliation: Chem. Dept., Virginia Polytech. Inst., Blacksburg, VA, USA
Journal: Analytica Chimica Acta, Computer Techniques and Optimization vol.103, no.4 p.459-68
Publication Date: 15 Dec. 1979 Country of Publication: Netherlands
CODEN: CTOPD3 ISSN: 0378-4304
Conference Title: Computers and Optimization in Analytical Chemistry
Conference Date: 5-7 April 1978 Conference Location: Amsterdam, Netherlands
Language: English
Abstract: This permits program preparation and data manipulation/storage/plotting/printing to take place on the CPU most appropriate to the task. It also allows the application programs to run on satellite processors with minimum configuration and maximum data throughput capabilities. The hardware implementation is described, as well as a brief introduction to the software facilities available under RT-11/REMOTE and FORTH.
Subfile: A C

62/3,AB/4 (Item 1 from file: 350)
DIALOG(R)File 350:Derwent WPIX

013514674
WPI Acc No: 2000-686620/200067
XRPX Acc No: N00-507671
Configurable processor designing system for processors, generates **hardware implementation description** and corresponding software development tools based on **configuration** specification of **processors**
Patent Assignee: TENSILICA INC (TENS-N)
Inventor: DIXIT A B; GONZALEZ R E; KILLIAN E A; LAM M; LICHTENSTEIN W D; MAYDEN D E; ROWEN C; RUDELL R; RUTTENBERG J; TJIANG W K; WANG A R; WILSON R P; MAYDAN D E
Number of Countries: 090 Number of Patents: 003
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
WO 200046704	A2	20000810	WO 2000US3091	A	20000204	200067	B
AU 200034841	A	20000825	AU 200034841	A	20000204	200067	
EP 1159693	A2	20011205	EP 2000913380	A	20000204	200203	
			WO 2000US3091	A	20000204		

Priority Applications (No Type Date): US 99322735 A 19990528; US 99246047 A 19990205; US 99323161 A 19990527
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
WO 200046704	A2	E	215	G06F-017/50	
	Designated States (National): AE AL AM AT AU AZ BA BB BG BR BY CA CH CN CR CU CZ DE DK DM EE ES FI GB GD GE GH GM HR HU ID IL IN IS JP KE KG KP KR KZ LC LK LR LS LT LU LV MA MD MG MK MN MW MX NO NZ PL PT RO RU SD SE SG SI SK SL TJ TM TR TT UA UG UZ VN YU ZA ZW				
	Designated States (Regional): AT BE CH CY DE DK EA ES FI FR GB GH GM GR IE IT KE LS LU MC MW NL OA PT SD SE SL SZ TZ UG ZW				
AU 200034841	A			G06F-017/50	Based on patent WO 200046704
EP 1159693	A2	E		G06F-017/50	Based on patent WO 200046704
	Designated States (Regional): AL AT BE CH CY DE DK ES FI FR GB GR IE IT LI LT LU LV MC MK NL PT RO SE SI				

Abstract (Basic): WO 200046704 A2
Abstract (Basic):

NOVELTY - Based on **configuration** specification of a **processor**, a **description** of a **hardware implementation** of the processor is generated. Software development tools specific to the **hardware implementation** is generated depending on **configuration** specification of the **processor**. The software development tools include a compiler, assembler, linker, disassembler, debugger and an instruction set simulator.

DETAILED DESCRIPTION - An INDEPENDENT CLAIM is also included for **configurable processor** designing method.

USE - For designing **configurable processor** in multiprocessor systems.

ADVANTAGE - Since directories only contain description of new enhancements and not the entire software system, the storage space required is minimal. By providing a constrained domain of extensions and optimizations, the process can be automated to a high degree thereby facilitating fast and reliable development. By providing **hardware description** language **description** of circuitry necessary to implement the instruction set, and development tools such as compiler, assembler, debugger and simulator which can be used to generate software for processor and to verify the processor are generated with less area, power consumption and speed.

DESCRIPTION OF DRAWING(S) - The figure shows the block diagram showing the flow of **processor configuration**.

pp; 215 DwgNo 6/24

EE TIMES ONLINE

www.cmpnet.com
The Technology Network

ESC: Chess/Checkers set to play for application-specific DSP develop

By Peter Clarke, EE Times
Nov 2, 1998 (6:54 AM)
URL: http://www.eetimes.com/story/OEG19981102S0002

LEUVEN, Belgium — Target Compiler Technologies NV will introduce its Chess/Checkers retargetable development environment for application-specific DSPs at the Embedded Systems Conference this week. The start-up, a spinoff from the Interuniversities Microelectronics Center (IMEC) research organization, has been working on the development since its founding in 1996. With the tools, the company claims, designers will be able to use C-language code to drive the exploration of different instruction sets and architectures, rather than select an established core and then write software for that core. The tools will let designers recompile their code for a number of custom architectures and run simulations before selecting and synthesizing an application-specific, DSP-like processor.

And, according to Target, because application-specific instruction processors can be matched precisely to requirements, they can cut IC power consumption by up to 90 percent and die area by half, compared with conventional general-purpose solutions.

The Chess/Checkers tools have been undergoing alpha and beta test with a number of customers since 1997. Motorola has designed and programmed a new application-specific processor for GSM speech coding using the tools, the company said.

The Chess/Checkers environment consists of a C compiler called Chess, a linker called Bridge, an instruction-set-simulator called Checkers and an assembler and disassembler called Darts. In contrast with other DSP support tools, Chess/Checkers is a retargetable environment.

Designers can specify the behavior of their application-specific DSP in a high-level processor description language designated nML.

Low-level code Once an nML file specifies the target processor, Chess reads the file to generate low-level code from C-language software. Similarly, Checkers uses the file to accept the low-level code and simulate its running on the target processor.

"Until recently, tool support was only considered after the microprocessor architecture was finished," said Gert Goossens, general manager of Target. "With Chess/Checkers, tools are available right from the beginning of the architectural design process. Designers can perform true architectural exploration by trying out alternative architectures in nML and evaluating their performance by running the retargetable compiler and instruction-set simulator."

While most of Target's re-targetable technology comes from IMEC, a first version of the nML language was developed at the Technical University of Berlin. "Experience shows that nML is quickly accepted by architecture designers and programmers who are used to writing assembly code,"

Goossens said.

"The early use of the tools during the architecture-design phase naturally leads to an architecture that is well-suited to the Chess compiler," he said. "The compiler also uses novel algorithmic optimizations that improve the code quality. As a result, large applications can be implemented completely from C specs and the resulting code quality is comparable to manually optimized code."

Chess provides specific support for DSPs that feature heavily encoded instructions and a heterogeneous register set. The compiler assumes time-stationary coding and that every instruction executes in a single cycle. This allows the processing pipeline to be fully controlled by the software that is generated by the Chess compiler and reduces the complexity of DSP control logic.

EETIMES
December 08, 1997, Issue: 984
Section: News

Target's tools let applications software drive synthesis -- Startup shoots for app-specific DSPs

Peter Clarke

Leuven, Belgium - Startup Target Compiler Technologies NV says it's within weeks of launching a set of Unix-based high-level system-design tools that represent a step up in design abstraction and turn at least one aspect of traditional digital-signal-processor-based ASIC development on its head.

With the tools, designers will be able to use application software to drive the exploration of different instruction sets and architectures, rather than selecting an established core that represents the best compromise among various projected performance parameters and then writing software for that core. The tools will let designers recompile their code for a number of custom architectures and run simulations before selecting and synthesizing an application-specific DSP-like processor.

Because application-specific instruction processors (ASIPs) can be developed to match a particular embedded-processing requirement precisely, they can cut IC power consumption by up to 90 percent and die area by half in such embedded applications as telecommunications and consumer electronics, according to Target, which was spun off from the Interuniversities Microelectronics Center (IMEC; Leuven)

"Users want application-specific processors because the performance is much better," said Target general manager Gert Goossens. "There are examples where an ASIP design uses a factor of 10 less power. But [ASIP design] is rarely done, because there is no tool support."

It is that lack of support that Target claims to have addressed with its Chess/Checkers design environment.

The software has been available in beta release since June. Goossens said Motorola Semiconductor Product Sector's wireless-communications center (Toulouse, France), an early customer, is using the software to develop a new DSP core for GSM mobile-telephone handsets. That's an application, Goossens said, where "you want the efficiency of custom

hardware but you still need programmability; therefore you still need a DSP."

Retargetable elements

The Chess/Checkers environment is comprised of three main elements: the Chess C-language compiler, the Checkers instruction-set simulator and a processor-modeling language called nML. Goossens said Chess and Checkers are retargetable, meaning they can quickly be ported to an alternative ASIP architecture. According to Target documentation, systems or processor designers will be able to perform the adaptation without the aid of Target staff.

The nML language enables the retargetability. Once an nML file specifies the target processor, Chess reads the file to generate low-level code from C-language software. Similarly, Checkers uses the file to accept the low-level code and simulate its running on the target processor.

"We developed the nML language to model DSP," said Goossens. "It's at a much higher level than Verilog or VHDL. It's at the level of a programmer's manual-something that could be used by assembler writers."

Alternative instruction-set architectures can be described in a few days' work, and their performance can be compared by running Chess and Checkers with modules of typical or critical code. Only after optimizing the instruction-set architecture is the translation made from nML to a hardware-description language such as VHDL or Verilog.

Although nML is essentially a behavioral language, Goossens said, "there is some structural information. You can specify whether multiple multipliers are supported, the memory structure, the cache type and so on."

Chess, Checkers and nML support the full range of traditional arithmetic, logical and control instructions. "We haven't applied Chess/Checkers to full-blown microprocessor design so far. That's not what it was developed for, and it might be difficult to do," said Goossens. "The main limitation is that we only support time-stationary coding; that is, we assume every instruction executes in a single cycle."

An architecture can still have pipelining of instruction fetches, decoding and execution. But deeper pipelines or execution over multiple cycles-as are often found in RISC microprocessors-are not supported. "Single-cycle execution is a valid assumption when you are creating your own architecture," Goossens said.

He said Target chose to craft Chess, the retargetable compiler, for the C language because"C is the de facto standard." The startup has no plans to address more object-oriented programming languages, such as C++ or Java, though it does "allow some features from C++ to be supported. For example, we allow user-specified data types; it's not just 32-bit data types" that are supported.

To demonstrate the utility of the C compiler, Target has applied it to the Analog Devices AD2100 16-bit fixed-point DSP. Users thus have a point of reference against which to benchmark their own instruction sets and architectures, Goossens said.

HDL generator

To complement Chess and Checkers, Target is developing a tool that would automatically translate nML code into synthesizable VHDL. That product is slated for introduction by the end of 1998.

"The first version of the HDL generator will be VHDL. We are collaborating with a local company, Easics, on the development of that," Goossens said. "Verilog will be important, and we should be able to structure it so that a Verilog version will be available only a few months later."

There are also plans to develop additional analysis and diagnostic tools to sit alongside the basic Chess/Checkers design flow.

"The nML language is a formal way to explore ideas," Goossens said. "The idea is to stay at the nML level until late in the design." He cited the importance of such elements as early and accurate power estimation, statistical information about the utilization of the architecture, and diagnostic information on speed and code density.

Links to third-party tools "are necessary," Goossens said. "Checkers should be able to work in a cosimulation mode."

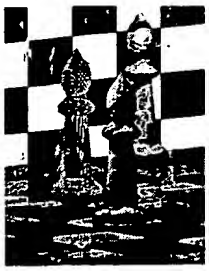
Indeed, Target has already provided links so that Checkers can work with the Eagle-i simulator, from Viewlogic/Synopsys.

"Our part of the cooperation has been done," Goossens said. Some extensions are still being worked on at the other end to allow Eagle-i to work with DSP simulators.

Target is just over a year old and has a staff of only five. Nonetheless, Chess and Checkers come to market with a pedigree: The tools were developed by Goossens and colleagues at IMEC, Europe's leading independent research organization for silicon process technology and EDA-tool development. Goossens, Werner Guerts, Johan Van Praet and Dirk Lanneer, the four founders of Target, all transferred from IMEC with the technology.

Over the past decade, IMEC has pioneered a number of high-level design tools that have been offered to industry. DSP-synthesis tools DSPstation and Mistral, for example, were transferred to Silvar-Lisco and then to what became the European Development Center of Mentor Graphics before being passed to their current owner, Frontier Design NV (Leuven)

Other tools developed at IMEC include hardware/software-codesign offerings commercialized by CoWare NV.



Home
Company
Products
News
Jobs
Contact

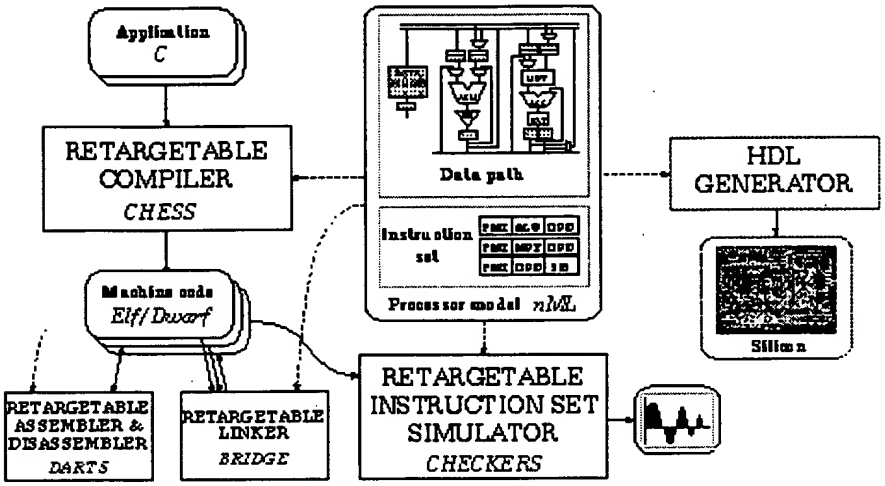
TARGET
COMPILER TECHNOLOGIES

Technical Brief

The Chess/Checkers retargetable DSP environment

Chess/Checkers is a computer-aided design environment, intended for developers and programmers of application-specific digital signal processor (DSP) cores.

Chess/Checkers is a *retargetable environment*. The designer can specify the characteristics of the target DSP in a high-level processor description language. Automatically the Chess/Checkers tools will work for this newly specified DSP. The processor description language is called *nML*.

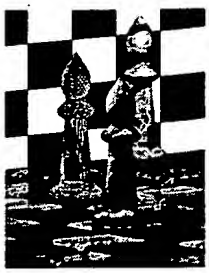


Chess/Checkers consists of the following tools:

- Chess is a *software compiler* that maps *C* application programs into highly optimised machine code for the target DSP. Chess can cope well with architectural peculiarities of DSPs, such as: specialised arithmetic instructions, a large variety of data types, specialised address generation units, heterogeneous register structures, and various degrees of instruction encoding (ranging from VLIW to highly encoded instruction sets). Chess produces machine code in the *Elf/Dwarf* object file format.
- Checkers is an *instruction-set simulator* that simulates the execution of machine code in a cycle and bit accurate way. The simulator can either be executed as a stand-alone tool, or be coupled to an existing hardware description language (HDL) simulator, thereby enabling co-simulation of a processor in its environment. Checkers supports *C* source level debugging based on *Elf/Dwarf* executable files.
- Bridge is a *linker* that builds an executable file from separately compiled *Elf/Dwarf* object files for different *C* functions.
- Darts is an *assembler* and *disassembler* that translates machine code from assembly into binary format and back.

webmaster@retarget.com

Last updated : Thu Sep 10 1998.



Home
Company
Products
News
Jobs
Contact

TARGET
COMPILER TECHNOLOGIES

Fact Sheet

Chess - Retargetable Compiler

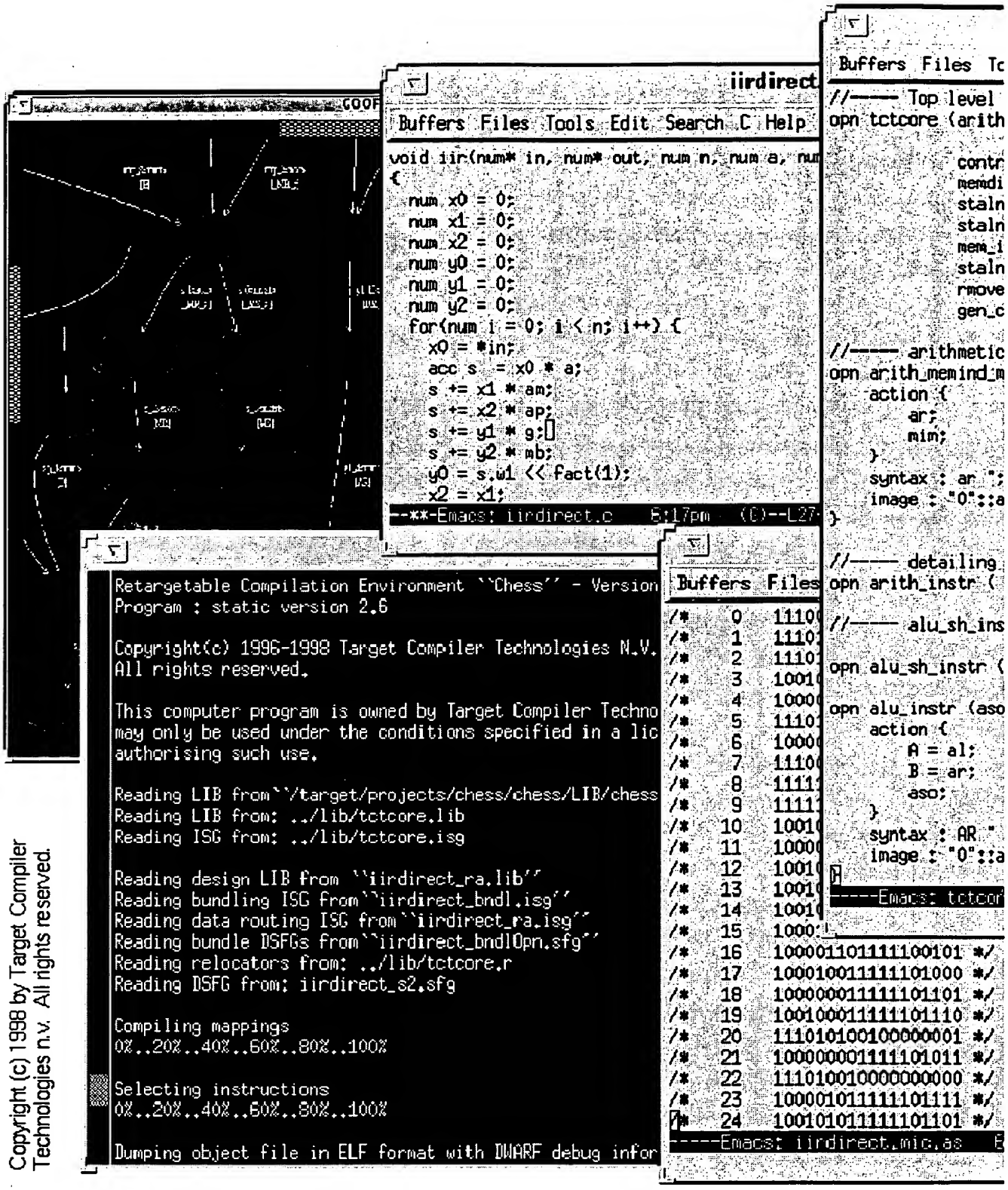
Screen dump (40.3K image)

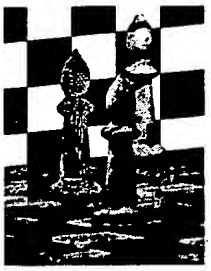
Chess is a retargetable software compiler for DSPs. It offers the following features:

- A unique approach to *compiler retargetability*, based on the processor description language *nML*. *nML* is a high-level language at the abstraction level of a programmer's manual.
- Capabilities for *architectural exploration*. Users can describe alternative DSP architectures in *nML*, and compare their performance by compiling benchmark *C* function onto each architecture and evaluating the results.
- Support for a wide range of *DSP architectures*, with the following main characteristics:
  - General as well as customised arithmetic instructions and data types;
  - A wide variety of instruction formats, ranging from highly encoded to orthogonal (``long instruction word'') formats;
  - A preference for load-store architectures;
  - Register structures ranging from general-purpose register files to special-purpose registers;
  - Time-stationary instructions, i.e. all instructions must complete their execution in a single machine cycle.
- Application programming in *C*, with the additional support of the *C++* concepts of classes and operator overloading. This allows for the specification and use of *user-defined data types*.
- Efficient *compiler optimisations*:
  - A large set of high-level code optimisations, including: data-flow analysis, constant folding, strength reduction, common sub-expression elimination, induction variable analysis, loop-invariant code analysis, hardware loop selection, dead and unreachable code elimination, in-line expansion;
  - Code selection, exploiting the use of specialised instructions such as multiply-accumulate patterns, indirect addressing with pointer post-modification, etc.;
  - Register allocation of intermediate variables, supporting many different routing schemes to carry data between storage elements;
  - Scheduling to exploit the DSP's parallelism, including the optimisation of software pipelining.
- Support for subroutines and interrupt routines based on a software stack in data memory.

- Generation of binary machine code in the standard object file format *Elf*.
- Generation of *source-level debugging* information in *Elf* object files (using *Dwarf-2* sections). This information can be used by instruction-set simulators such as Checkers or by hardware debugging environments, to enable source-level debugging during the execution of the generated code.
- The Chess/Checkers environment is currently available on Solaris and HPUX Unix platforms.

webmaster@retarget.com
Last updated : Thu Sep 10 1998.



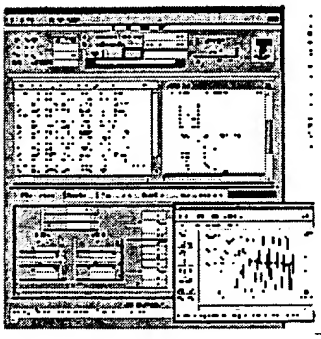


Fact Sheet

Checkers - Retargetable Instruction-Set Simulator

Home
Company
Products
News
Jobs
Contact

TARGET
COMPILER TECHNOLOGIES

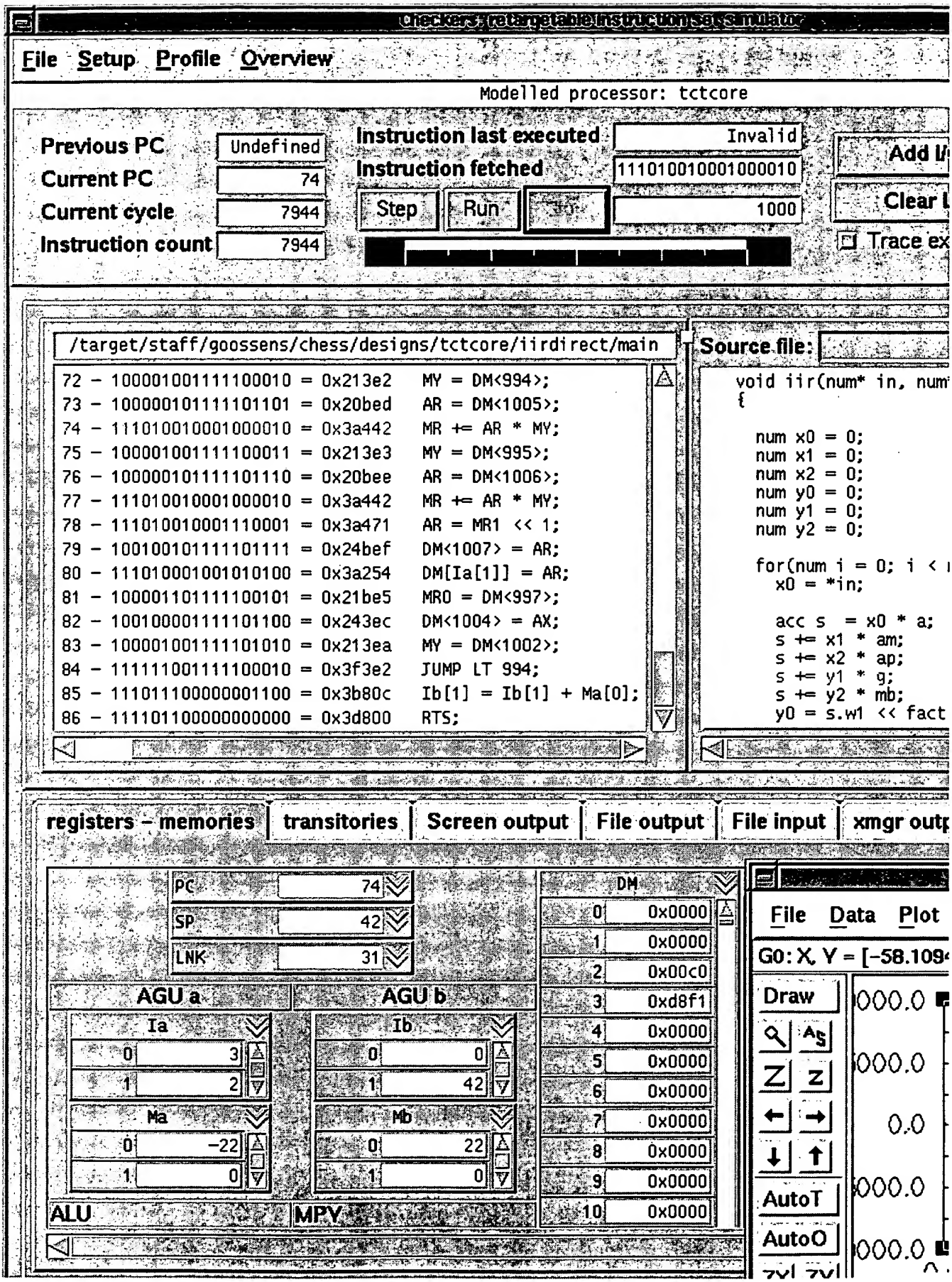


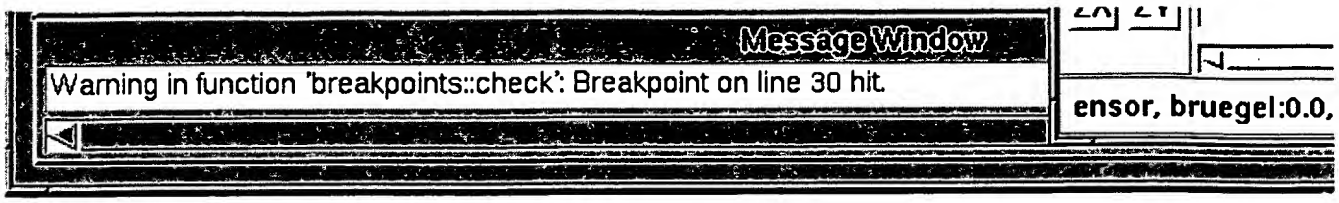
Screen dump (56K image)

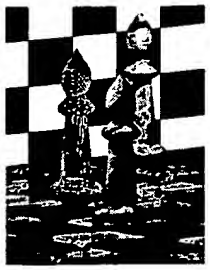
Checkers is a retargetable instruction-set simulator (ISS) for DSPs. It offers the following features:

- A unique approach to *ISS retargetability*, based on the processor description language *nML*. *nML* is a high-level language at the abstraction level of a programmer's manual. Together with the *nML* description of the DSP, the user provides bit-accurate simulation models of the DSP's primitive operations, written in *C++*.
- Efficient *cycle-* and *bit-accurate simulation* of the execution of machine code on the target DSP. Simulation speed is comparable to high-level *C* simulation (in the order of 100,000 instructions per second for a typical DSP, on an HP B/132L desktop workstation).
- Loading of executable machine code files in the standard file format *Elf*, optionally containing source-level debug information in *Dwarf-2* format, as generated by the *Chess* compiler.
- User control via the command line, or via an easily customisable graphical user-interface (GUI) with the following features:
  - Multi-functional application code window, displaying binary machine code, assembly code and *C* source code. *C* source code display requires the use of *Elf* executable files with *Dwarf-2* sections. During simulation, the executed instructions and corresponding *C* statements are highlighted.
  - Display of data values in memory locations, registers and connections.
- Easy interfacing to existing HDL simulators and to third-party co-simulation tools, to model the DSP in its environment.
- Binding of memory locations to data files on the host computer's file system, to read input stimuli from files and write output results to files.
- Support of breakpoints on instructions and on C statements. Support of watchpoints on storage locations. Profiling of instructions.
- Plotting of data values in storage locations as waveforms, and post-processing of waveforms.
- The Chess/Checkers environment is currently available on Solaris and HPUX Unix platforms.

webmaster@retarget.com
Last updated : Thu Sep 10 1998.


Checkers: retargetable instruction set simulator
File Setup Profile Overview
Modelled processor: tctcore
Previous PC
Undefined
Current PC
74
Current cycle
7944
Instruction count
7944
Instruction last executed
Invalid
Instruction fetched
11101001000100010
Step
Run
1000
Add
Clear
Trace ex
/target/staff/goossens/chess/designs/tctcore/iirdirect/main
72 - 100001001111100010 = 0x213e2 MY = DM<994>;
73 - 100000101111101101 = 0x20bed AR = DM<1005>;
74 - 111010010001000010 = 0x3a442 MR += AR * MY;
75 - 100001001111100011 = 0x213e3 MY = DM<995>;
76 - 100000101111101110 = 0x20bee AR = DM<1006>;
77 - 111010010001000010 = 0x3a442 MR += AR * MY;
78 - 111010010001110001 = 0x3a471 AR = MR1 << 1;
79 - 100100101111101111 = 0x24bef DM<1007> = AR;
80 - 111010001001010100 = 0x3a254 DM[Ia[1]] = AR;
81 - 100001101111100101 = 0x21be5 MR0 = DM<997>;
82 - 100100001111101100 = 0x243ec DM<1004> = AX;
83 - 100001001111101010 = 0x213ea MY = DM<1002>;
84 - 111111001111100010 = 0x3f3e2 JUMP LT 994;
85 - 111011100000001100 = 0x3b80c Ib[1] = Ib[1] + Ma[0];
86 - 111101100000000000 = 0x3d800 RTS;
Source file:
void iir(num* in, num
{
num x0 = 0;
num x1 = 0;
num x2 = 0;
num y0 = 0;
num y1 = 0;
num y2 = 0;
for(num i = 0; i <
x0 = *in;
acc s = x0 * a;
s += x1 * am;
s += x2 * ap;
s += y1 * g;
s += y2 * mb;
y0 = s.w1 << fact
registers - memories
transitories
Screen output
File output
File input
xmgr outp
PC 74
SP 42
LNK 31
AGU a
AGU b
Ia
0 3
1 2
Ib
0 0
1 42
Ma
0 -22
1 0
Mb
0 22
1 0
ALU
MPY
DM
0 0x0000
1 0x0000
2 0x00c0
3 0xd8f1
4 0x0000
5 0x0000
6 0x0000
7 0x0000
8 0x0000
9 0x0000
10 0x0000
File Data Plot
G0: X, Y = [-58.109
Draw
AutoT
AutoO
000.0
000.0
0.0
000.0
000.0


Message Window
Warning in function 'breakpoints::check': Breakpoint on line 30 hit.
ensor, bruegel:0.0,



Fact Sheet

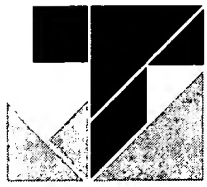
Bridge & Darts - Retargetable Back-End Tools

Bridge is a retargetable linker that builds an executable file from separately compiled *Elf/Dwarf-2* object files for different *C* functions.

Darts is a retargetable assembler and disassembler that translates machine code from assembly into binary format and back. The assembly language syntax can be defined by the user, and is specified as part of the *nML* description of the DSP.

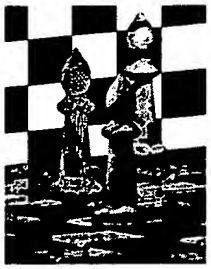
The Chess/Checkers environment is currently available on Solaris and HPUX Unix platforms.

Home
Company
Products
News
Jobs
Contact



TARGET
COMPILER TECHNOLOGIES

webmaster@retarget.com
Last updated : Thu Sep 10 1998.



Press Release

For immediate release

Target Compiler Technologies Unveils Retargetable Support Tools for Application-Specific DSPs

* * *

Home
Company
Products
News
Jobs
Contact

TARGET
COMPILER TECHNOLOGIES

New `Chess/Checkers' Environment to be Demonstrated at 1998 Embedded Systems Conference

Leuven, Belgium - October 19, 1998.

Target Compiler Technologies (`Target'), a manufacturer of advanced EDA tools based in Leuven, Belgium, today announced that it will introduce its new tool suite, called Chess/Checkers, at the upcoming Embedded Systems Conference. The Chess/Checkers environment supports the design and use of application-specific digital signal processors.

The new environment has been under development by Target since its spin-off from the Belgian micro-electronics research centre IMEC, mid 1996. The tools have been under alpha and beta test at selected customers' sites, since 1997. The Chess/Checkers tools have been successfully applied in design projects by first customers, including the design and programming of a new application-specific processor for GSM speech coding by Motorola.

The Chess/Checkers environment consists of a C compiler (Chess), a linker (Bridge), an instruction-set simulator (Checkers), and an assembler and disassembler (Darts). In contrast to many other DSP support tools, Chess/Checkers is a retargetable environment. Designers can specify the behaviour of their application-specific DSP in a high-level processor description language, called `nML'. Automatically all tools will work for the specified processor architecture.

The Chess/Checkers tools are primarily intended for designers of application-specific DSP cores in high-volume markets, such as telecom and consumer applications. By optimising the DSP core's architecture and instruction set to the application at hand, superior performance and a reduced footprint and power dissipation are obtained compared to general-purpose cores. The tools provide a powerful infrastructure for embedded systems companies, allowing to create, maintain and use new proprietary IP blocks in the form of application-specific DSP cores, independent from external processor vendors.

According to Gert Goossens, Target's general manager, Chess/Checkers turns the traditional approach to processor design on its head. ``Until recently, tool support was only considered after the processor architecture was finished,'' Goossens said. ``With Chess/Checkers, tools are available right from the beginning of the architecture design process. Designers can perform true architectural exploration, by trying out alternative architectures in nML and evaluating their performance by running the retargetable compiler and instruction-set simulator,'' he added.

nML is a high-level language that captures a programmer's model of the processor. A first version of the language was developed at the Technical University of Berlin. Target said it significantly modified the language to make it well suited for retargetable compilation. ``Experience shows that nML is quickly accepted by architecture designers and programmers who are used to writing assembly code," Goossens said.

Target currently offers Chess/Checkers as a retargetable tool set, allowing users to enter and modify their own nML processor models. Target also said that it will be able to create non-retargetable versions of the tools, incorporating a fixed nML model provided by its customers. These non-retargetable tools can be distributed to system integrators who want to implement their system on the application-specific processor.

According to Goossens, existing design projects have shown that the Chess compiler is able to produce efficient code. ``The early use of the tools during the architecture design phase naturally leads to an architecture that is well suited for the Chess compiler," he said. ``The compiler also uses novel algorithmic optimisation techniques that improve the code quality. As a result, large applications can be implemented completely from C specs and the resulting code quality is comparable to manually optimised assembly code."

Chess favours a certain style of DSP architectures. It provides specific support for DSPs featuring heavily encoded instructions and a heterogeneous register set. The compiler assumes time-stationary coding, i.e. every instruction executes in a single cycle. This implies that the data pipeline is fully controlled by the software that is generated by the Chess compiler. This significantly reduces the complexity of the DSP's controller.

Chess/Checkers is currently available on Unix platforms. The tools will be demonstrated at the 1998 Embedded Systems Conference, San Jose Convention Center, booth T408, San Jose, California, November 3-5, 1998.

Corporate Information

Target Compiler Technologies n.v. is an innovation company, specialising in design technologies for embedded software in electronic systems. Target was incorporated in 1996 as a spin-off company of IMEC, the Belgian R&D centre for micro-electronics. Target's shareholders are IMEC v.z.w., Software Holding & Finance n.v. (a Belgian private holding, investing in high-tech companies) and Tetracom b.v.b.a. (a group representing Target's management). Target is managed by Gert Goossens, Dirk Lanneer, Werner Geurts and Johan Van Praet, who all transferred from IMEC in 1996.

Target focuses on system design companies in competitive market segments like telecommunications and consumer electronics. Target's main product, Chess/Checkers, is based on a unique, patented technology initiated at IMEC.

Contact

Gert Goossens

Target Compiler Technologies N.V.
Phone: +32-16-40 81 14
Fax: +32-16-40 53 00
E-mail: goossens@retarget.com
WWW: http://www.retarget.com

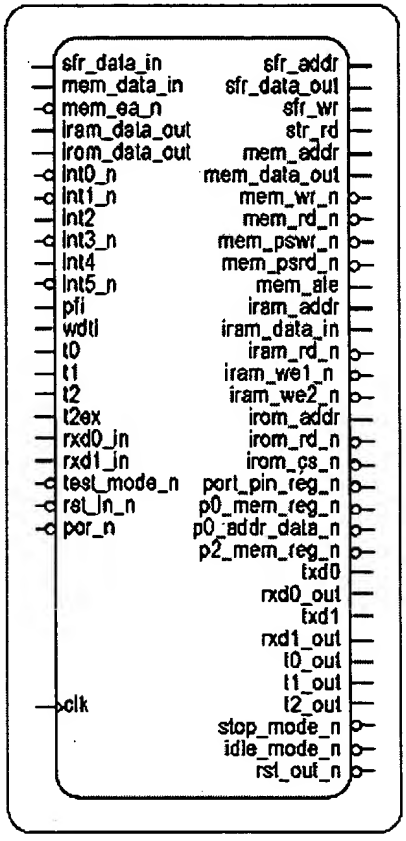
Last updated : Mon Oct 19 1998.

DesignWare DW8051 MacroCell

The synthesizable DesignWare DW8051 MacroCell provides two-and-a-half times the performance of standard 8051 microcontrollers.

Overview

The DesignWare(R) DW8051(TM) MacroCell is a high-performance, configurable, fully-synthesizable, and reusable 8051 core. It is fully binary compatible with the industry standard 803x/805x microcontrollers and is available in either Verilog or VHDL source.



DW8051 Input/Output Signals

High Performance and Portability

The DesignWare DW8051 MacroCell solution includes automatic synthesis and test scripts, a reference design, and our extensive verification environment. The

DW8051's high-performance architecture provides up to two-and-a-half times the performance improvement over the standard 8051 when operating at the same clock rate.

It synthesizes to run at 50 megahertz in typical 0.5-micron processes, and up to 90 megahertz in 0.35-micron processes.

The DW8051 is technology independent, so it can be implemented in a variety of process technologies. The DW8051 has been proven in both FPGA and CBA ASIC technologies.

Proven Quality, Complete Solution

To ensure quality, the DW8051 was developed according to Synopsys' strict design-for-reuse methodology. The DW8051 has undergone extensive testing during the design process, has been ported to several different technologies, and has been fabricated in CBA technology to ensure both performance and compatibility. DW8051 CBA-based chip samples are available for evaluation. The DW8051 has also been tested with a variety of third-party 8051 development tools and 8051 evaluation boards. The DW8051's high-performance, configurable, synthesizable architecture, along with the development environment provided and supported by Synopsys, provide a total solution for building low-cost, high-performance embedded control systems for a wide range of applications.

Technical Advantages of the DW8051

- Four clocks/instruction cycle versus 12 in standard 8051
  - Typically 2.5 times faster execution versus standard 8051
- Stretch memory cycle
  - Allows application software to adjust to different external RAM speeds
  - MOVX in as little as 8 clock cycles
- Dual data pointers
  - Improves efficiency when moving large blocks of data
- Internal/external peripheral interface
  - Special function register (SFR) bus in DW8051 supports both internal and external peripherals versus internal only in standard 8051
- Two optional full-duplex serial ports
- Seven additional interrupts

DW8051 Features

The DW8051 MacroCell is provided in VHDL or Verilog source. It is reusable in design environments that include widely used EDA tools for simulation (e.g., VCS (TM), VHDL System Simulator(TM) (VSS), Vantage, MTI, Leapfrog, and Verilog-XL), Synopsys Design Compiler(TM) for synthesis, and Synopsys Test Compiler (TM) for test.

803x/805x Compatibility
The DW8051 is compatible with the standard 8051 instruction set and can be configured to a wide range of industry standard 803x/805x architectures.

Control signals for standard 803x/805x I/O ports are included. Optional full-duplex serial ports and third timer are selectable through parameters.

High-Performance Architecture
DW8051's design is fully static and synchronous. Greater efficiency and performance is achieved by eliminating wasted bus cycles, and by providing dual data pointers for moving large data blocks and a 16-bit address memory interface. The DW8051 core is typically 10k-13k gates depending on configuration and the technology it is implemented in. It runs from 0 megahertz up to 90 megahertz (90-megahertz operation requires a target technology of 0.3 micron or less). Lower performance applications also benefit by being able to run at lower clock rates to get the same performance as a standard 12-cycle instruction 8051. Lower clock rates lead to lower power consumption and electro-magnetic interference (EMI).

Flexible Peripheral Interfaces
With the DW8051, you can configure the internal RAM and ROM sizes. Its well-defined SFR interface allows for straightforward integration of user defined peripherals. Variable length MOVX accesses fast and slow peripherals.

Third-Party Development Tools Support
Synopsys has an active program in place to support third-party tools. Many industry standard compilers, assemblers, ROM monitors, and in-circuit emulators have been tested for compatibility with the DW8051. This allows integration of these tools into a design environment and provides a complete development solution for DW8051-based embedded systems on a chip. In-circuit emulation support is provided by Nohau Corporation and Hitex Development Tools.

DW8051 Configurable Architecture

Figure 1 illustrates the hardware architecture of the DW8051 core. The name of the top-level module is DW8051_core. The internal RAM and ROM modules are located outside DW8051_core to facilitate simulation and insertion of technology-specific

RAM/ROM modules. The following submodules and interfaces are optional and selectable through parameter settings:

- DW8051_core can address either 128 or 256 bytes of internal RAM
- The internal ROM address range is determined by a parameter rom_addr_size
- Timer 2 (DW8051_timer2) is optional
- 0, 1, or 2 serial ports (DW8051_serial) can be implemented
- The interrupt unit is either DW8051_intr_0 (6-source) or DW8051_intr_1 (13-source)

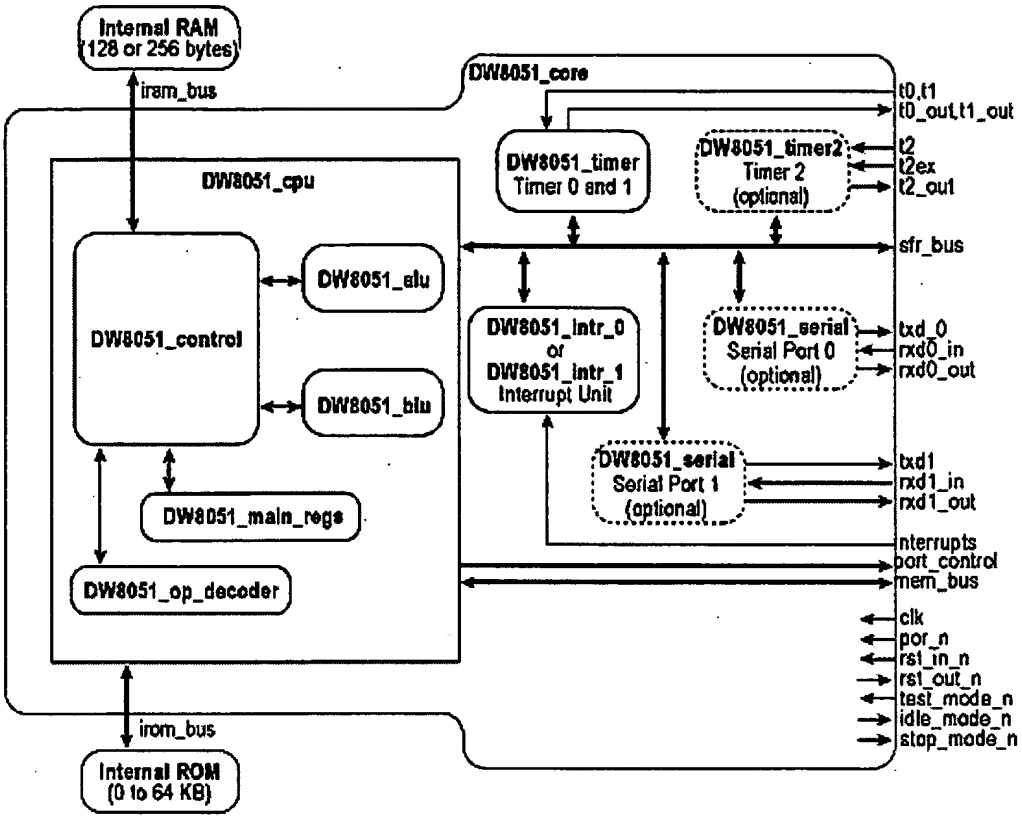


Figure 1: DW8051_core

803x/805x Feature Comparison

Through parameter settings, you can configure the DW8051 hardware to be functionally compatible with a variety of 803x/805x configurations. For example, you can implement two 16-bit timers for compatibility with the Intel 8051, or you can implement three for compatibility with the Intel 80C32. Table 1 provides a feature-by-feature comparison of the DW8051 MacroCell and several common 803x/805x configurations.

Feature	Intel				Dallas	DesignWare
	8031	8051	80C32	80C52	DS80C320	DW8051
Clocks Per Instruction Cycle	12	12	12	12	4	4
Internal ROM (1)	—	4KB	—	8KB	—	Up to 64KB
Internal RAM (1)	128 bytes	128 bytes	256 bytes	256 bytes	256 bytes	128 bytes or 256 bytes
Data Pointers	1	1	1	1	2	2
Serial Ports	1	1	1	1	2	0, 1, or 2
16-bit Timers	2	2	3	3	3	2 or 3
Interrupt Sources (total of int. and ext.)	5	5	6	6	13	6 or 13
Stretch Memory Cycle	No	No	No	No	Yes	Yes

(1) Internal ROM and RAM are located outside of DW8051_core.

Table 1: Feature Summary of DW8051 and Common 803x/805/x Configurations

Performance Overview

The DW8051 processor core offers increased performance by executing instructions in a 4-clock bus cycle, as opposed to the 12-clock bus cycle in the standard 8051. The shortened bus timing improves the instruction execution rate for most instructions by a factor of three over the standard 8051 architectures.

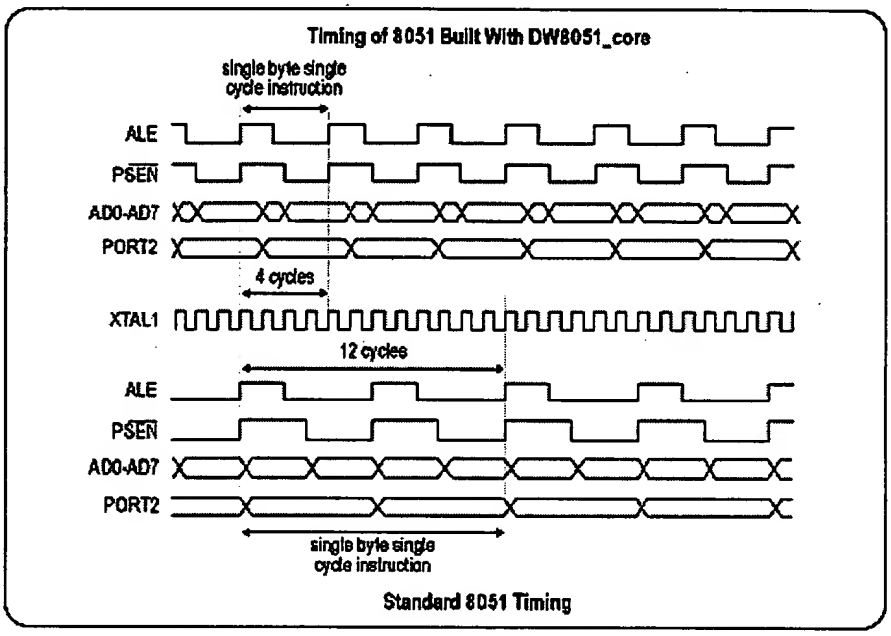


Figure 2: Instruction Cycle Timing Comparison

Some instructions require a different number of instruction cycles on the DW8051 than they do on the standard 8051. In the standard 8051, all instructions except for MUL and DIV take one or two instruction cycles to complete. In the DW8051 architecture, instructions can take between one and five instruction cycles to complete. The average speed improvement for the entire instruction set is approximately two-and-a-half times, calculated as seen in Table 2.

Number of Opcodes	Speed Improvement
150	3.0X
51	1.5X
43	2.0X
2	2.4X

Note: Comparison is for DW8051 and standard 8051 operating at the same clock frequency.

Table 2: Performance Comparison of DW8051 vs. Standard 8051

DW8051 Development Environment

The DW8051 MacroCell includes:

- VHDL/Verilog source code for the DW8051 MacroCell
- Multiple-simulator support (e.g., VCS, VSS, Vantage, MTI, Leapfrog, Verilog-XL), Synopsys synthesis, and Synopsys Test Compiler support
- An example 8032-compatible design
  - This design uses the DW8051_core and illustrates how to build and connect 8051-compatible port modules for designs where it is preferable to use standard 8051 port modules instead of the high-speed memory interface
- Extensive verification environment
  - HDL testbench that instantiates the DW8051_core, models internal ROM and RAM, and emulates 64 kilobytes of external RAM and 64 kilobytes of external ROM.
  - Processes that trace the program counter and write accesses to external RAM.
  - A collection of 8051 assembler programs that test all of the instruction set opcodes, plus miscellaneous tests for internal hardware. These tests are executed directly by the testbench.
  - A set of expected results (golden files) and a script that compares your simulation results against the expected results.
- A complete set of Synopsys synthesis scripts
  - These can be used to compile a specific implementation to your target technology.
  - Full-scan test insertion scripts for Synopsys Test Compiler are also included.
- Complete documentation
  - DW8051 databook in hardcopy and on-line format.
- Support for third-party development tools
  - Industry standard compilers, assemblers debuggers, ROM monitors, in-circuit emulators from Nohau and Hitex.
- Technical support
- Evaluation board and CBA silicon samples available

Embedded System Design with DW8051

As illustrated in Figure 3, Synopsys provides a complete solution for developing your embedded system-on-a-chip design with the DW8051 MacroCell. For more information, contact your local local sales office.

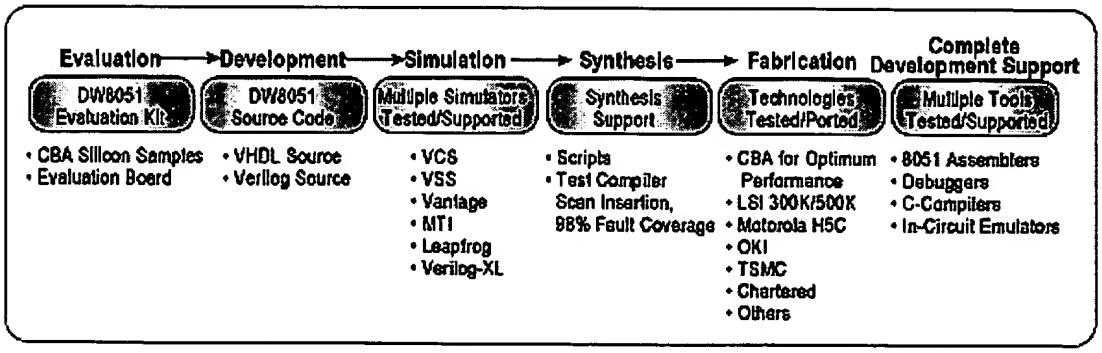
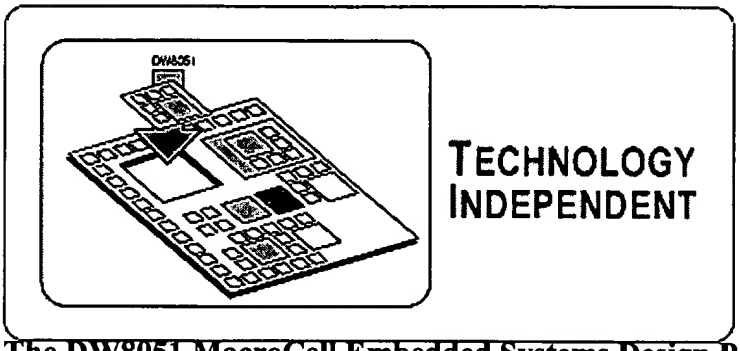
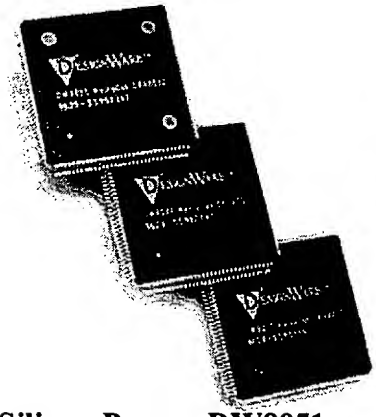


Figure 3: Embedded System Design with DW8051



The DW8051 MacroCell Embedded Systems Design Package

Hitex ICE Supports the DW8051



Silicon-Proven DW8051

Nohau Real Time Microprocessor Development Tools Support the DW8051

For pricing information, please contact your local sales office.
For more information, complete the product information form.

DesignWare DWPCI MacroCell Solution

The DWPCI is an easy-to-use, synthesizable, high-performance core solution that is ideal for all PCI interface applications.

Overview

The Synopsys DesignWare(R) DWPCI(TM) MacroCell is a high-performance, configurable, synthesizable PCI core specifically designed for fast, easy integration into your ASIC design. The DWPCI MacroCell solution includes a unique automated user interface that guides you through installation, configuration and synthesis. The combination of a robust, silicon-proven design, user configurability, automated synthesis and a full verification suite makes this the highest quality, easiest-to-use PCI solution available.

Complete Solution

To provide a complete solution for PCI design, the DWPCI MacroCell solution includes:

- Configurability to support a variety of PCI 2.2-compliant design implementations including:
  - 64- or 32-bit PCI bus width
  - 66 or 0-33 MHz PCI clock (with full-scan test insertion)
- Power management interface (PMI) support
- Vital product data (VPD) register support
- VHDL or Verilog source code
- Automated user interface for installation, configuration and synthesis
- Extensive and flexible verification environment
- Complete documentation package including extensive on-line help system
- Hot-line technical support

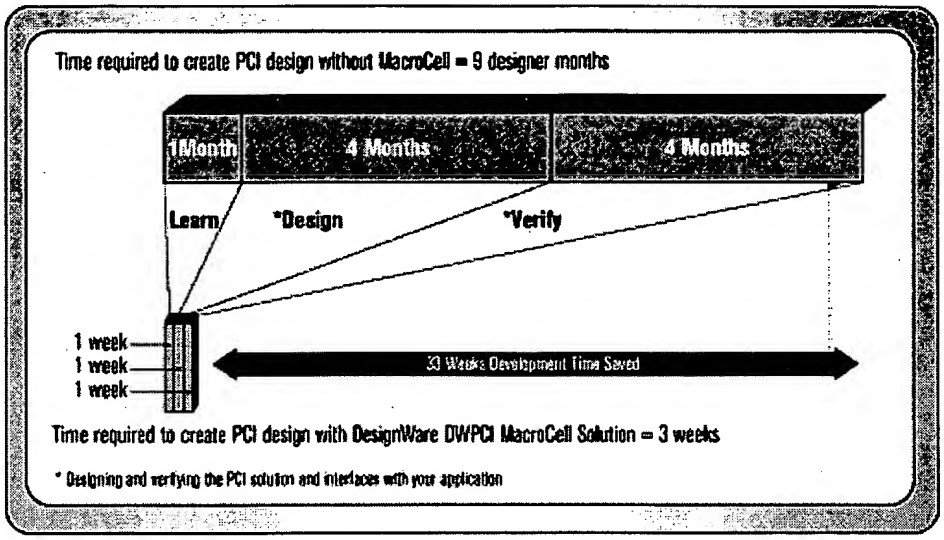


Figure 1: Time-to-Market Advantage of the DWPCI MacroCell Solution

Proven Quality

To ensure quality in every step of the DWPCI MacroCell development process, the DesignWare team followed strict engineering guidelines, including:

- Synopsys design-for-reuse coding guidelines and methodology
- Extensive testing at the module and system levels
- 100% HDL code coverage during module test
- Functional verification using the PCI 2.2 compliance test suite, plus comprehensive corner case and random tests
- Simulation on multiple vendors' logic simulators to ensure portability
- Synthesis in multiple configurations to technology libraries from multiple silicon vendors to ensure portability to your preferred technology
- Silicon-proven functionality and performance

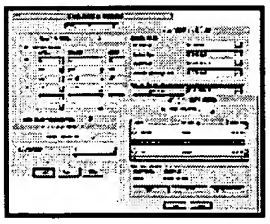


Figure 2: DesignWare DWPCI Automated User Interface Guides You Through Installation, Configuration and Synthesis of the DWPCI MacroCell

Easy to Use

The application interface, verification suite and automated user interface tool included in the DWPCI MacroCell solution combine to simplify your design flow. The application interface is synchronous to the PCI clock and uses easy-to- understand protocols (similar to PCI protocol) for target, initiator, and configuration space transactions. The verification suite includes master and slave bus functional models, a PCI SIG (Special Interest Group) compliance suite, and protocol checkers for both application and PCI interfaces to ensure compliance and successful integration into your application.

The automated user interface provides a completely new way of integrating intellectual property (IP) into your design. Through the graphical user interface (GUI) or from the command line interface, you install, configure, and synthesize the DWPCI MacroCell solution to meet your specific design requirements.

Installation - The user interface installs the DWPCI MacroCell then runs a basic set of simulation and synthesis verification tests to ensure that the DWPCI MacroCell is installed properly.

Configuration - The user interface guides you through setting parameters such as bus widths and base address register (BAR) values. It also checks that your selected parameter values and combinations of parameter values are valid, then automatically updates the HDL source code for your user configuration.

Synthesis - The user interface guides you through top-level timing constraint configuration, then it generates synthesis scripts optimized for your specific DWPCI MacroCell configuration and technology library. The user interface then invokes Synopsys Design Compiler (TM) to run the scripts and displays the resulting reports.

DWPCI MacroCell Design Features

- Complies with PCI Local Bus Specification, Version 2.1
- Parameterized PCI bus width and application address bus width
- Parameterized implementation of PCI configuration space and several PCI 2.1 and 2.2 optional features (see Table 1)
- Zero-wait-state PCI bus transactions
- Application interface that is easy to understand and integrate
- Supports PCI initiator and target operation
- Supports simultaneous configuration space transactions from PCI bus and the application bus
- Fully synchronous design
- Partitioned for best timing performance
- Partitioned along clear functional boundaries
- Automatic test insertion

PCI 2.1 Compliance Features

The DesignWare DWPCI MacroCell implements all of the features and

functionality required by PCI 2.1 and several of the optional features. Specific PCI 2.1 compliant features include:

- 66-MHz PCI clock in 0.35-micron (or smaller) technologies
- 0 to 33-MHz PCI clock in most target technologies
- 64- or 32-bit PCI AD bus
- PCI 2.1 type 00h configuration space header
- Dual address cycles
- Supports the PCI 16/8 clock rule
- Supports all PCI bus commands including Memory Write and Invalidate (MWI), interrupt acknowledge cycles, and special cycles
- Medium or slow DEVSEL# generation
- All types of abort, retry and disconnect
- Delayed transactions
- Fast back-to-back transactions
- Posted memory write
- Pre-fetching of memory read data from pre-fetchable address space
- VGA snooping (with fast, medium or slow DEVSEL# devices)
- Exclusive access (LOCK#) for target transactions
- Cache line wrap mode
- Initiator latency timer
- PCI interrupt pins
- PERR# and SERR#
- Expansion ROM BAR

PMI Compliance Features

The DWPCI MacroCell implements the functionality defined by the PMI 1.0 specification:

- PMI support is selectable through a parameter
- Supports PMI 66-MHz requirements
- Reset functionality of Power Management Control/Status Register (PMCSR), bits 15 and 8, is selectable through a parameter
- Supports all PMI states, including optional D1 and D2 states
- I/O and memory cycles are disabled and configuration cycles are enabled when in PMI states D1 and D2

PCI 2.2 ECR Support

The DWPCI MacroCell supports the proposed PCI 2.2 engineering changes requests (ECRs), including:

- Capabilities support (PCI SIG approved)
- Subvendor ID requirement (PCI SIG approved)
- VPD access method (PCI SIG approval pending)

These features are optional and selectable via the user interface.

Configurability

The DWPCI MacroCell is highly configurable to enable you to create a customized PCI MacroCell implementation that meets your design needs. Table 1 lists the configurable features of the DWPCI MacroCell. The user interface guides you through the configuration steps, then automatically updates the HDL code and synthesis scripts for your custom configuration.

DWPCI MacroCell Configurable Features	Options
PCI AD bus width	32 or 64 bits
Application address bus width	32 or 64 bits
DEVSEL# timing	Medium or slow
Cache line register size	0-8 bits
66 MHz Operation	Implemented or Not-implemented
PMI register block	Implemented or Not-implemented
VPD register block	Implemented or Not-implemented
Cache line wrap mode	Implemented or Not-implemented
Delayed transactions	Implemented or Not-implemented
Exclusive transactions	Implemented or Not-implemented
VGA palette snooping	Implemented or Not-implemented
Fixed CardBus card information structure (CIS) pointer	Fixed to a user-defined value or built as a programmable register
Vendor ID register	Fixed to a user-defined value
Device ID register	Fixed to a user-defined value
Subvendor ID register	Fixed to a user-defined value or built as a programmable register
Subsystem ID register	Fixed to a user-defined value or built as a programmable register
Revision ID register	Fixed to a user-defined value
Interrupt pin register	Fixed to a user-defined value or built as a programmable register
Minimum grant register	Fixed to a user-defined value or built as a programmable register
Maximum latency register	Fixed to a user-defined value or built as a programmable register
Latency timer register	Fixed to a user-defined value or built as a programmable register
Class code register	Fixed to a user-defined value
User-defined functions	Implemented, Not-implemented, or Programmable
Number of base address registers (BARs)	1-6 32-bit BARs (or 1-3 64-bit BARs)

For each BAR, the following attributes are parameterized: - Type of space - Type of access - Type of range - Number of bits needed to implement the address range	
Expansion ROM address range	0 or 8-21 address bits (up to 16 MB addressable)
Number of application-specific configuration space registers	0-48

Table 1: Configurable Features of the DWPCI MacroCell

Applications

Typical applications for the DWPCI MacroCell are in ASICs for add-in boards that connect to the PCI local bus, as illustrated in Figure 3. Such applications include LAN interfaces, mass storage device interfaces, graphics devices and multimedia devices.

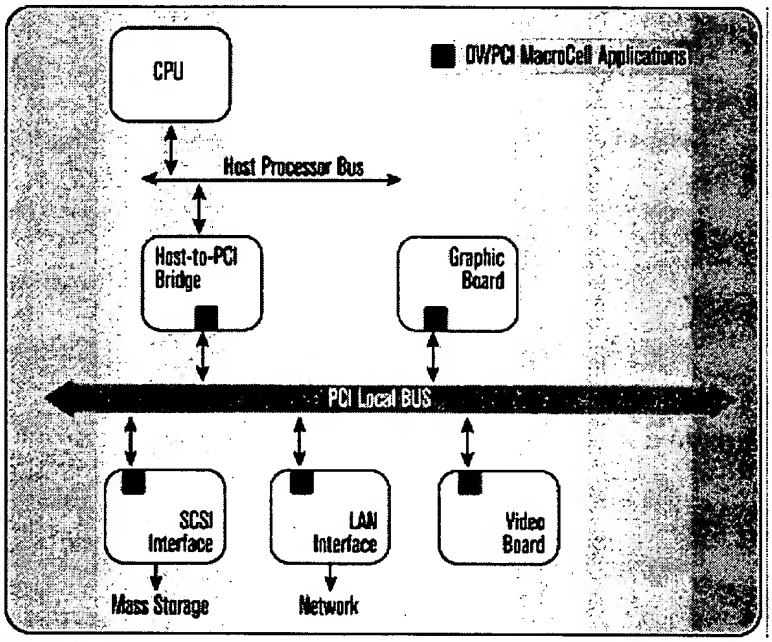


Figure 3: DesignWare DWPCI MacroCell Applications

Maximum Performance

The DWPCI MacroCell provides superior PCI bus performance with zero-wait-state data transfers. When configured for 64-bit PCI bus width, the DWPCI MacroCell provides a burst data transfer rate of 264 MB/s at 33

MHz, or 528 MB/s at 66 MHz.

PCI MacroCell Architecture

The DWPCI MacroCell (Figure 4) consists of a core (DWpci_core) plus an I/O pad module (DWpci_iop). The core implements the PCI protocol, configuration, and datapath functionality. The I/O pad module instantiates your selected silicon technology-specific I/O pads that connect DWpci_core to the PCI bus.

DWpci_core is partitioned for maximum performance, testability and functional clarity. The three modules that provide the PCI target functionality are grouped into the target block (DWpci_tar).

The three modules that provide the PCI initiator functionality are grouped into the initiator block (DWpci_initiator). The configuration block (DWpci_config) implements the PCI 2.1 type 00h configuration space. The interface block (DWpci_ifc) provides a common PCI interface for the target, initiator, and configuration blocks.

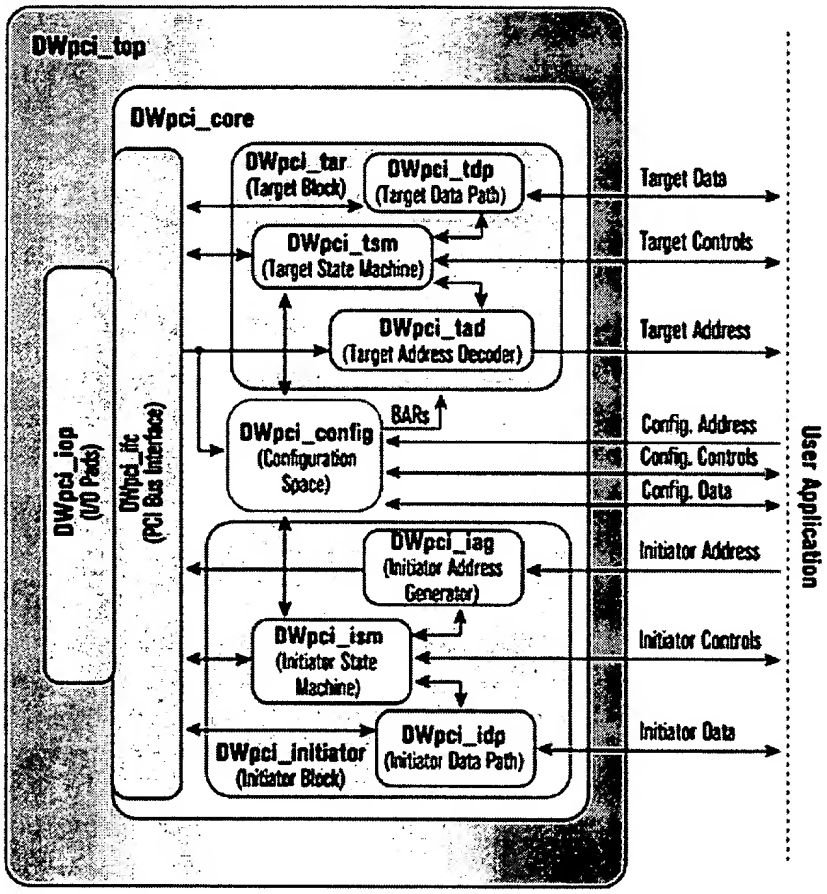


Figure 4: DWPCI MacroCell Block Diagram

Easy-to-Use Application Interface

The DWPCI MacroCell provides two interfaces: the PCI bus interface and the application interface. The PCI bus interface complies with the PCI Local Bus Specification, Rev. 2.1. The application interface includes a target application interface, an initiator application interface, and a configuration space interface. The application interface is synchronous to the PCI clock and is designed for easy integration of the DWPCI MacroCell into your application design.

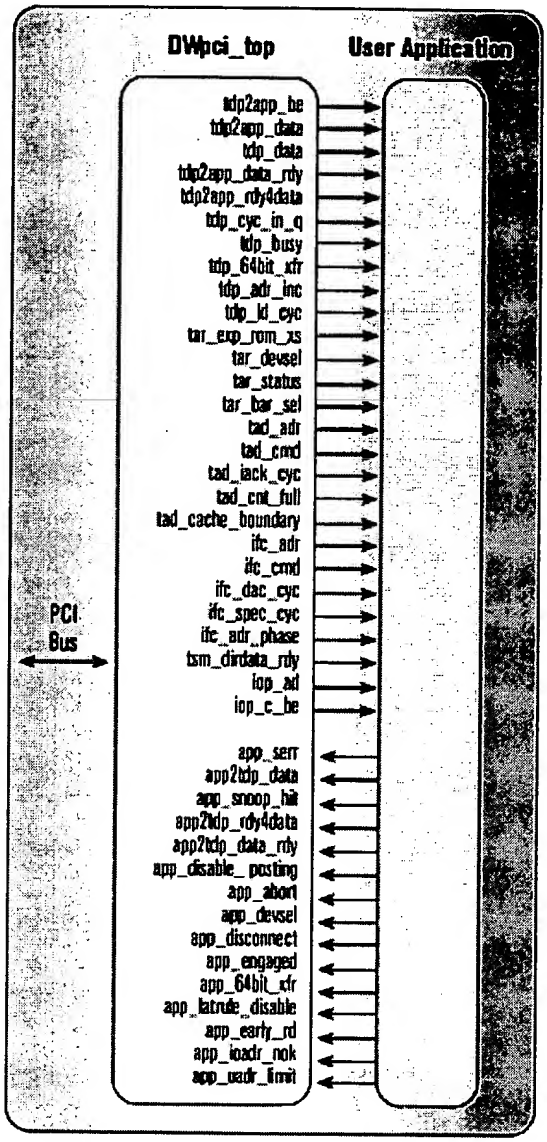


Figure 5: Target Application Interface

Target Interface

The target application interface (Figure 5) uses separate uni-directional address and data buses for read and write operations and a simple handshake protocol that is similar to the PCI bus protocol. Data transfers occur on the positive clock edge when both handshake signals are active. The DWPCI MacroCell provides several features that make the target interface easy to integrate into your application:

- The target interface uses a simple protocol that gives the application full control over posted and delayed transactions.
- The DWPCI MacroCell automatically checks for legal cache line size values and disconnects after the first data transfer if the cache line size value is illegal.
- The DWPCI MacroCell automatically disconnects a transaction when the target burst address crosses a BAR boundary.
- For pre-fetchable address regions where the application does not require the decoded target address, there is a simple protocol to execute an early (three clock cycle) read.
- The application has control over snooping and enabling/disabling the 16/8 clock latency timer through simple protocols.
- The target interface is flexible so that connection to the application can be direct or through a FIFO.

Initiator Interface

The initiator application interface (Figure 6) also uses separate uni-directional address and data buses for read and write operations and a simple handshake protocol that is similar to the PCI bus protocol. Data transfers occur on the positive clock edge when both handshake signals are active. The DWPCI MacroCell provides several features that make the initiator interface easy to integrate into your application:

- The application initiates a PCI bus cycle with a single strobe and does not need to re-request the bus in the event of a disconnect.
- The DWPCI MacroCell automatically manages most PCI protocol related activities (including retries, disconnects, and latency timer timeouts).
- The DWPCI MacroCell automatically terminates a MWI transaction at the cache line boundary in the event of a latency timer timeout.
- The initiator interface is flexible so that connection to the application can be direct or through a FIFO.

Configuration Space Interface

The application has access to DWPCI MacroCell configuration space registers through a configuration interface with a single address bus and uni-directional read and write data buses (Figure 7). For simultaneous accesses to configuration space from the PCI bus and from the application, the DWPCI MacroCell gives the PCI bus priority, allowing the PCI bus transaction to complete before performing the application-side access. In addition, the application has direct read access to the control, status, cache line size, and base address registers, and the VPD register flag and PMCSR.

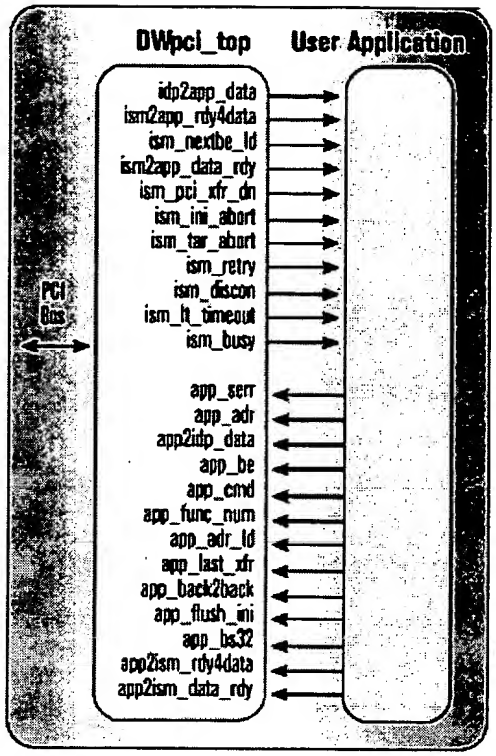


Figure 6: Initiator Application Interface

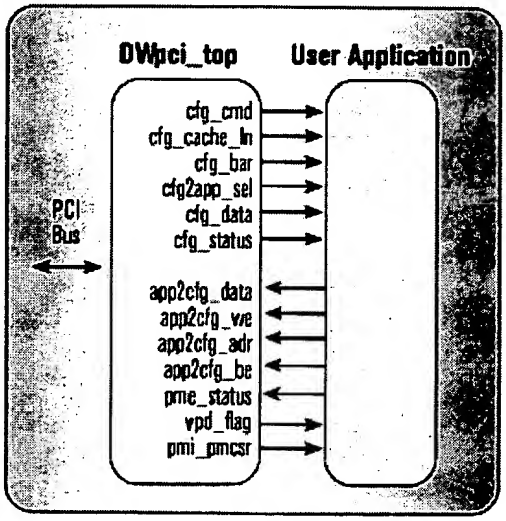


Figure 7: Configuration Space Application Interface

The DWPCI Verification Environment

The DWPCI MacroCell solution includes a complete verification environment, as shown in Figure 8. Bus functional models from Synopsys' Logic Modeling Group (LMG) provide a powerful mechanism for creating PCI bus transactions to and from the MacroCell. Using multiple master and

slave bus functional models, you can create complex, coordinated activity on the bus, simulating real system operation.

The DWPCI MacroCell solution also includes bus functional models for the application buses. The TAPP (target application) model responds to transactions on the target application bus, and the IAPP (initiator application) model initiates transactions on the initiator application bus.

All of these bus functional models can be controlled by separate command files or by a single command file. This feature enables you to create complex, coordinated test scenarios involving all of the bus functional models.

The DWPCI MacroCell solution also includes a set of bus monitors for verifying the transactions on the PCI and application buses. The PCIMONITOR performs full protocol checking on the PCI bus. The TMON (target bus monitor) provides protocol checking on the target application bus and the IMON (initiator bus monitor) provides protocol checking on the initiator application bus.The DesignWare development team used this full verification environment to perform a rigorous verification of the DWPCI MacroCell. This verification environment is included in the DWPCI MacroCell solution to make verification of the full ASIC complete, robust and easy to perform.

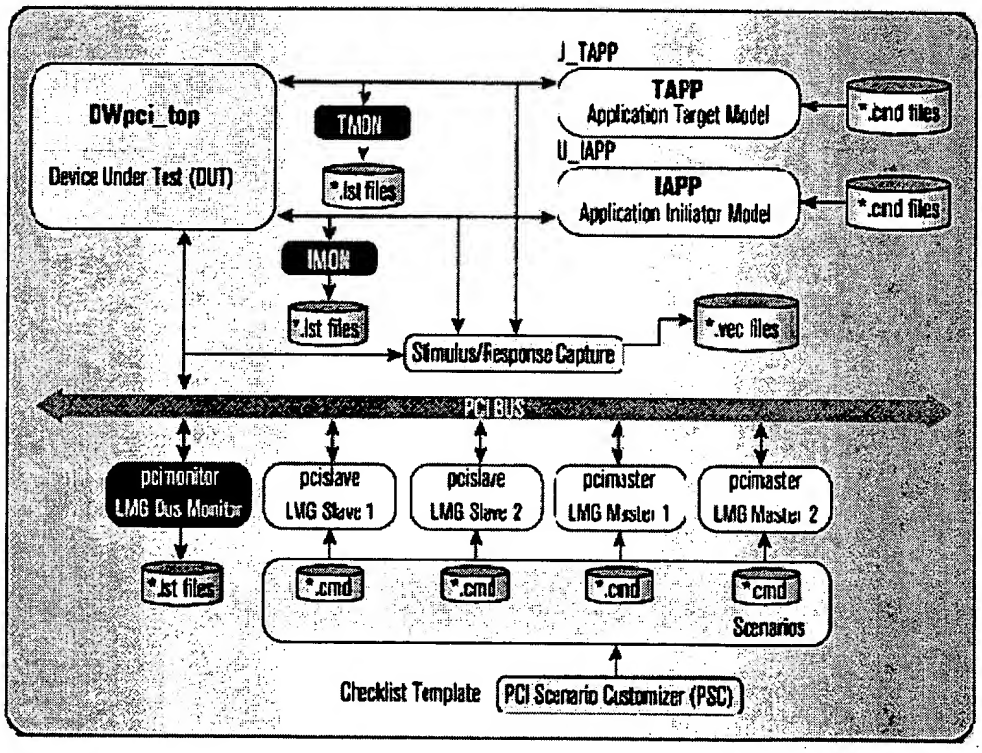


Figure 8: The DWPCI MacroCell Verification Environment

Automated User Interface Features

The DWPCI MacroCell's automated user interface is powerful and user-

friendly. It guides you through installation, configuration and implementation. You interact through either the GUI (Figure 9) or the command line interface to set parameters and timing constraints. The user interface automatically generates configured HDL code and synthesis scripts for you.

The GUI provides a top-level activity list that guides you through the DWPCI MacroCell design flow steps in the required order. The automated user interface presents the design flow steps in checklist format, with specific steps disabled until the prerequisite steps are completed. As you perform design activities, the user interface invokes design tools (for example, Design Compiler) to execute the design activities and updates the design files (source code and synthesis scripts) for your customized DWPCI MacroCell configuration.

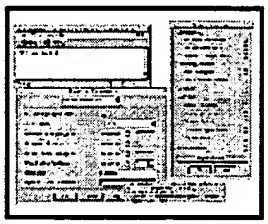


Figure 9: DesignWare DWPCI MacroCell's Automated User Interface

Throughout the design process, the automated user interface provides pop-up help tips, plus an extensive, hyperlinked on-line help system, with links to an HTML version of the DWPCI MacroCell Databook.

By providing these features and services, the user interface offers the following advantages over other PCI solutions:

- Quick and reliable installation, configuration and synthesis of the DWPCI MacroCell.
- Easy reconfiguration for subsequent designs, for example, to move from 32- to 64-bit PCI interface.

Installation
When you install the DWPCI MacroCell, the automated user interface prompts you for installation options. Based on these options, the user interface unpacks, analyzes and installs the design files into the proper directories.

To ensure that the installation was successful and that the DWPCI MacroCell will function properly in your environment, the user interface performs simulation verification with your selected simulation tool and synthesis verification with your selected version of Design Compiler.

Configuration
After you perform the initial DWPCI MacroCell installation, you can use the user interface to create one or more user-defined DWPCI MacroCell configurations. The user interface provides the following customization services:

- **Parameter setting** - The user interface prompts you for configuration information and sets MacroCell parameter values according to your answers. The user interface also checks that the parameter values are valid and combinations of parameter values are consistent with the PCI specification.

- **HDL generation** - After you select your parameter values, the user interface updates the HDL source code for your user configuration. You do not have to manually edit the MacroCell HDL source files.

Synthesis
The automated user interface provides the following synthesis services:

- **Technology library analysis** - The user interface analyzes your selected technology library to determine the library and cell characteristics, then uses this information for I/O pad insertion, technology-independent constraint specification, and synthesis script generation.

- **PCI I/O pad insertion** - The user interface prompts you to select technology-specific PCI bus I/O pads from your selected technology library and automatically instantiates these pads in the HDL source code.

- **Script generation and synthesis** - The user interface customizes a MacroCell synthesis strategy, based on Synopsys synthesis expertise and information gathered during technology library analysis. The user interface then prompts you to verify or change synthesis variables, strategy, and constraints. The constraints are initially specified using technology-independent formulas. After you accept the synthesis specifications, the user interface converts the constraint formulas into technology-specific constraint values and gener - ates custom synthesis scripts for your DWPCI MacroCell configuration and technology library. The user interface can invoke Design Compiler to execute the synthesis scripts or write them out to a file for later execution.

- **Synthesis results analysis** - The user interface reads the Design Compiler synthesis reports and presents the data to you in a hierarchy of PCI specific dialogs and hyperlinked HTML reports of the synthesis results.

Manufacturing Test Solution

An example Synopsys Test Compiler(TM) full-scan insertion script is provided with this solution. The fully synchronous design of the DWPCI MacroCell lends itself well to Test Compiler and other test insertion and vector generation tools. Full scan leads to very high fault coverage of the DWPCI MacroCell.

Conclusion

The DesignWare DWPCI MacroCell solution, with its robust verification environment, provides a complete PCI design solution that is customizable to meet your PCI design needs. Delivered with the powerful automated user interface, the DWPCI MacroCell offers a PCI design solution that not only provides silicon-proven PCI 2.1 compliance, performance and functionality, but is also unmatched in its ease of use.

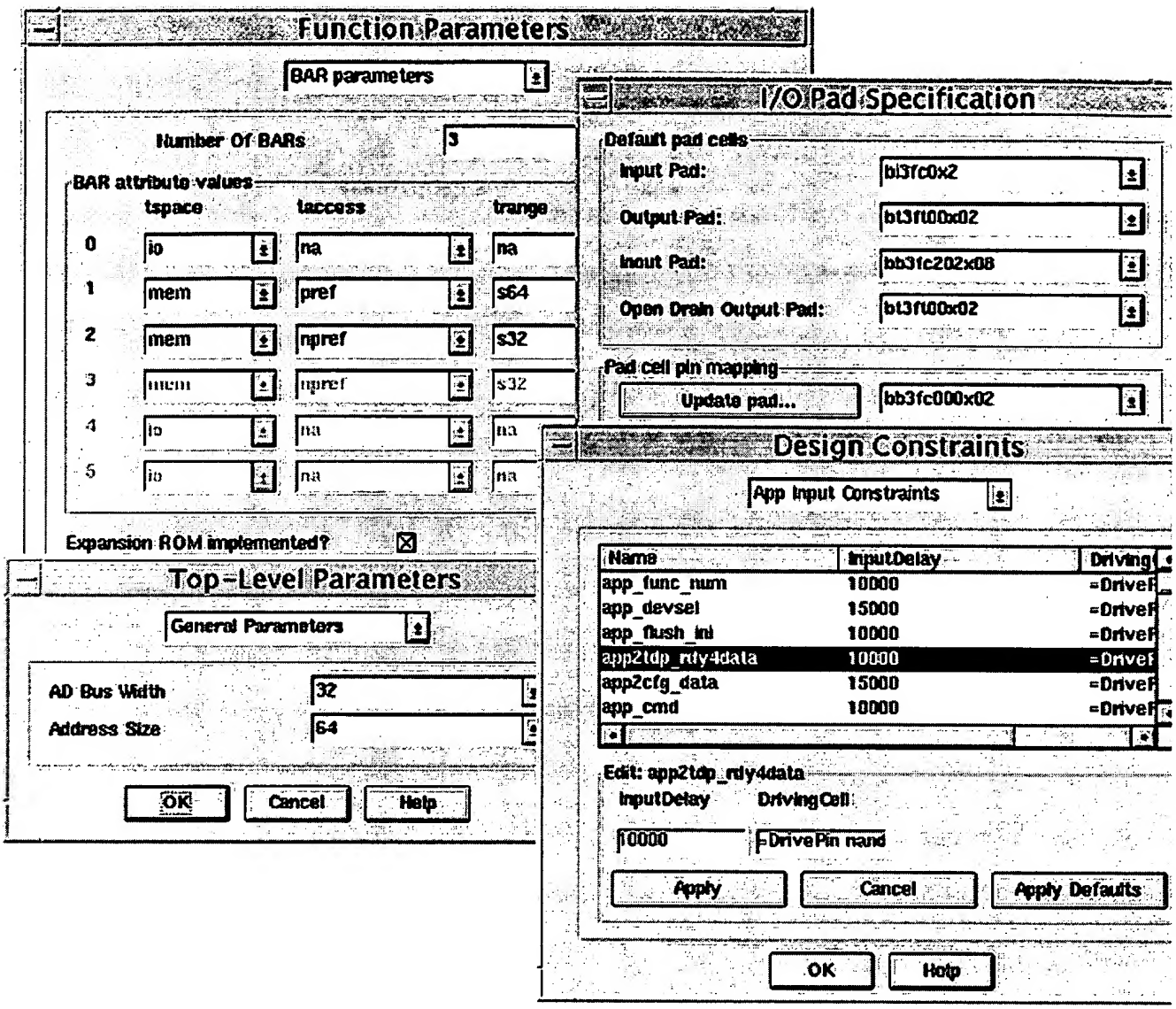
System Requirements

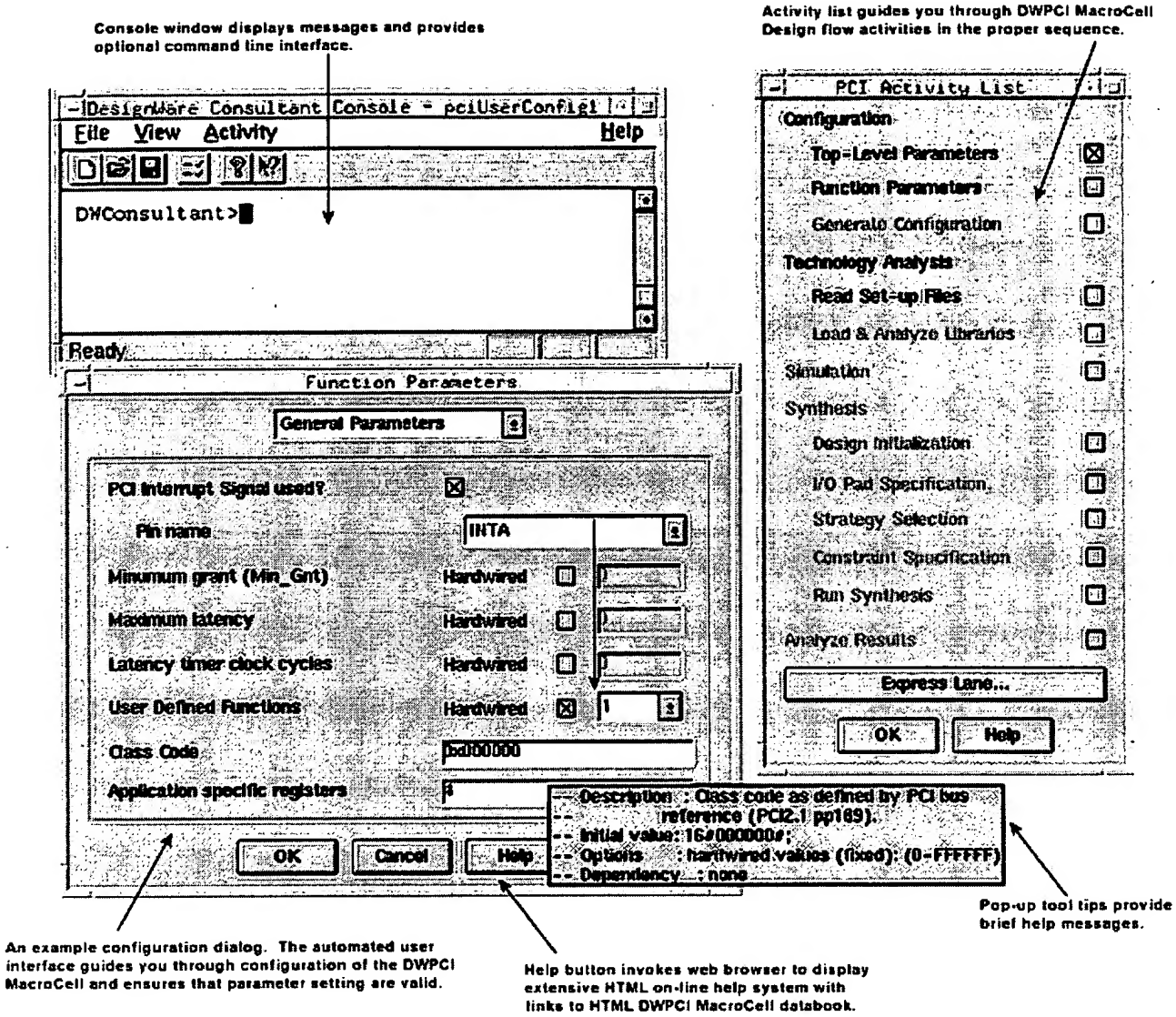
The Synopsys DesignWare DWPCI MacroCell solution requires the following resources:

- SPARC compatible workstation configured as follows:
  - Disk space-490 MB
  - Swap space-1.2 GB
  - Physical memory-128 MB
  - CD-ROM drive
- SunOS 4.1.3 and Solaris 5.5.1 platforms supported
- Synopsys Design Compiler and Synopsys (V)HDL Compiler, version 1997.08 or later
- VHDL or Verilog simulator
- Technology library files, including PCI I/O pads from your ASIC manufacturer. Synopsys supplies a 0.35-micron CBA (Cell-Based Array) library as an example.

For pricing information, please contact your local sales office.
For more information, complete the product information form.

| Home | Search | Help | Feedback | Site Map |


Function Parameters
BAR parameters
Number Of BARs
3
BAR attribute values
tspace
taccess
trange
0 io na na
1 mem pref s64
2 mem npref s32
3 mem npref s32
4 io na na
5 io na na
Expansion ROM implemented?
I/O Pad Specification
Default pad cells
Input Pad:
bi3fc0x2
Output Pad:
bt3ft00x02
Inout Pad:
bb3fc202x08
Open Drain Output Pad:
bt3ft00x02
Pad cell pin mapping
Update pad...
bb3fc000x02
Design Constraints
App Input Constraints
Name
InputDelay
Driving
app_func_num 10000 =DriveF
app_devsel 15000 =DriveF
app_flush_ini 10000 =DriveF
app2tdp_rdy4data 10000 =DriveF
app2cfg_data 15000 =DriveF
app_cmd 10000 =DriveF
Edit: app2tdp_rdy4data
InputDelay
DrivingCell
10000
=DrivePin nand
Apply
Cancel
Apply Defaults
OK
Help
Top-Level Parameters
General Parameters
AD Bus Width
32
Address Size
64
OK
Cancel
Help


Console window displays messages and provides optional command line interface.
Activity list guides you through DWPCI MacroCell Design flow activities in the proper sequence.
DesignWare Consultant Console - pciUserConfig1
File View Activity Help
DWConsultant>
Ready
PCI Activity List
Configuration
Top-Level Parameters
Function Parameters
Generate Configuration
Technology Analysis
Read Set-up Files
Load & Analyze Libraries
Simulation
Synthesis
Design Initialization
I/O Pad Specification
Strategy Selection
Constraint Specification
Run Synthesis
Analyze Results
Express Lane...
OK
Help
Function Parameters
General Parameters
PCI Interrupt Signal used?
Pin name
INTA
Minumum grant (Min_Gnt)
Maximum latency
Latency timer clock cycles
User Defined Functions
Class Code
0x000000
Application specific registers
Hardwired
OK
Cancel
Help
-- Description : Class code as defined by PCI bus reference (PCI2.1 pp169).
-- Initial value: 16#000000#;
-- Options : hardwired values (fixed): (0-FFFFFF)
-- Dependency : none
An example configuration dialog. The automated user interface guides you through configuration of the DWPCI MacroCell and ensures that parameter setting are valid.
Help button invokes web browser to display extensive HTML on-line help system with links to HTML DWPCI MacroCell databook.
Pop-up tool tips provide brief help messages.

Lexra is developing two product lines: The LX-4000 series of RISC processors and the LX-5000 series of DSPs. Following is a brief summary of the initial products in each line.

LX-4080 High Performance 32-Bit Embedded Processor

The LX-4080 provides the optimal price/performance value for embedded applications, whether price is judged in terms of silicon die size or power dissipation. The LX-4080 delivers 133 Dhrystone 2.1 MIPs, operates at 100 MHz, and occupies less than 2 mm^2 in die area and consumes less than 250 mW on a .35μ process. The LX-4080 is available today in either sythesizable RTL or as SmoothCore™ (hard macro).

LX-4080P 32-Bit Embedded Processor for Programmable Logic Devices

The LX-4080P is Lexra's LX-4080 RISC architecture, optimized for the Altera Flex 10KE® device family. The LX-4080P operates at 33 MHz and occupies less than 50% of the available gates in a Flex10KE PLD.

LX-5080 32-Bit Fixed-Point Digital Signal Processor

The LX-5080 is scheduled for production in 2H '99. Lexra's development team understands the use of DSPs in today's applications, and has the advantage of developing an architecture from scratch. The LX-5080 is poised to be an architectural breakthrough in ease of use, functionality, and performance.

HIGH PERFORMANCE LX-4080

32-Bit Embedded Processor
Product Brief

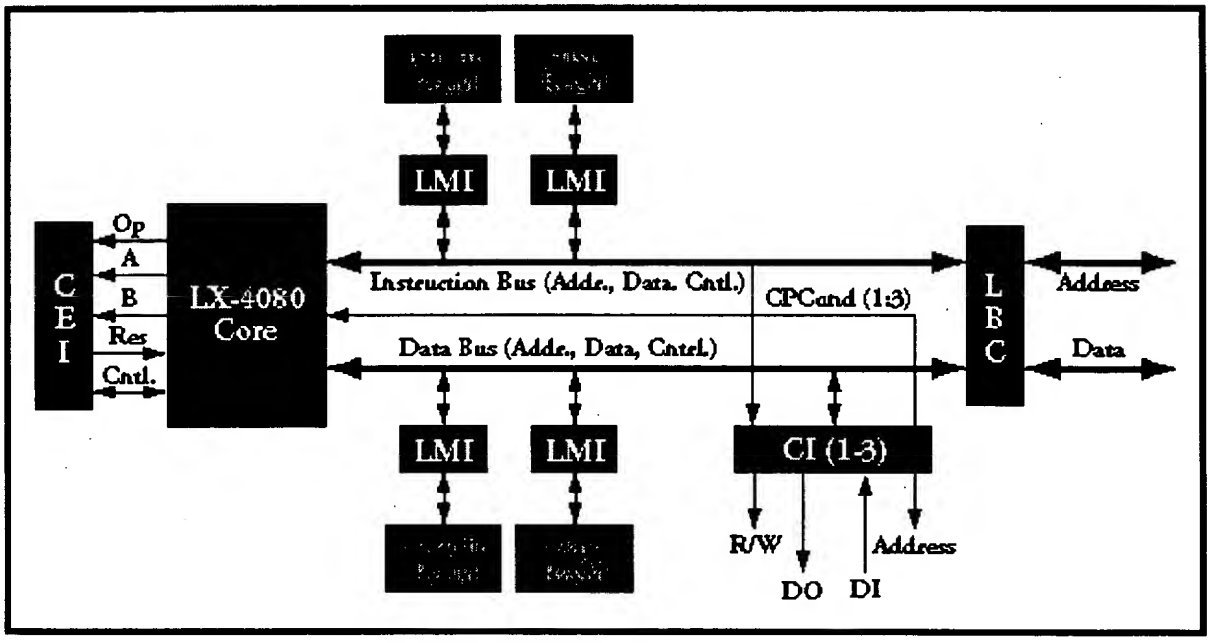
- **Best Price/Performance:** 100 MHz performance in less than 2 mm^2 on a 0.35μ process. Up to 133 Dhrystone 2.1 MIPs.
- **Supports MIPS-I® Instruction Set Architecture*:** Minimizes software porting effort and leverages broad avialability of software development tools. *Supports all MIPS-I instructions except unaligned loads and stores, which are emulated in software.
- **Easy ASIC Design:** Single phase clocking, fully synchronous design, easy to interface system bus protocol. Support for popular EDA tools.
- **Portability:** Available in RTL form, or in SmoothCore? form for popular COT foundries and ASIC vendors.
- **Easy Customization:** User-defined cache, RAM, ROM and address space sizes, plus an interface for adding application-specific custom instructions.

MIPS, R3000, R4000 and any other MIPS common law marks are tradmarks and/or registered trademarks of MIPS Technologies, Inc. Lexra, Inc. is not associated with MIPS Technologies, Inc. in any way.

Overview

The LX-4080 is an embedded RISC processor core targeted at system-on-silicon designers who require the best in price/performance and ease-of-use. Key applications include digital communications and consumer products such as network protocol processors, cable modems, set-top boxes, printers, next-generation disk controllers, and digital cameras.

In these high-volume products, cost and time-to-market are the two most critical issues for a design team. The LX-4080 addresses both concerns. With a die size of 2 mm^2, the LX-4080 is substantially smaller than most MIPS® R3000-class solutions, and with133 Dhrystone 2.1 MIPs, the LX-4080 delivers enough performance so designers do not have to resort to more expensive processors or cores. The LX-4080's time-to-market advantage is a result of Lexra's simple system level interface, complete set of software development tools, easy hardware design methodology, and flexible silicon manufacturing options.



Architecture Overview

The LX-4080 is a 32-bit R3000-class RISC processor. Lexra's VLSI architecture is quite different from that of other R3000-class processors and processor cores, and delivers much higher performance in a smaller silicon area. Lexra has taken advantage of the latest advances in design methodology and deep submicron process technology, and has designed the LX-4080's clocking, pipeline structure, pin-out, and memory interfaces to reflect system-on-silicon design needs.

MIPS-I Instruction Set Architecture*: The LX-4080 supports the MIPS-I programming model. The instruction set incorporates a three-port register file. Two source operands can be supplied and one destination updated per cycle. The second operand is either a register or 16-bit immediate. The instruction set includes a wide selection of ALU operations executed by the RALU, Lexra's proprietary register based ALU. The RALU also generates memory addresses for 8-bit, 16-bit and 32-bit register loads from (stores to) memory by adding a register base to an immediate offset. Branches are based on comparisons between registers, rather than flags, and are therefore easy to relocate. Optional links following jump or branch instructions assist with subroutine programming.

*The MIPS unaligned load and store instructions are supported through software emulation only. Because these instructions are seldom used in embedded applications, a hardware implementation represents a poor price/performance tradeoff. With softare emulation, the absence of hardware support for these instructions does not affect the software programming model.

Pipeline: Instructions are executed by a five-stage pipeline, which is designed so that all transactions internal to the LX-4080 and at the LX-4080 interfaces occur on the positive edge of the processor clock. Two-phase clocks are not used. As a result, designs based on the LX-4080 will achieve higher clock speeds in deep submicron technologies without requiring expensive Phase Locked Loop (PLL) logic.

Exception Handling: The LX-4080 supports the MIPS-I exception handling model. Exceptions include instruction-synchronous *traps* and hardware and software *interrupts*. All exceptions are prioritized. When an exception is taken, control is transferred to a user program located at the exception vector. The use program identifies the cause of the exception and transfers control to the application-specific handler.

Coprocessor Operations: The LX-4080 supports all 32-bit Coprocessor operations. These include moves to and from the Coprocessor general registers and control registers (MTCz, MFCz, CTCz, CFCz), Coprocessor loads and stores (LWCz, SWCz) and branches based on Coprocessor condition flags (BCzT, BCzF). The Lexra-supplied Coprocessor Interface can support all Coprocessor Operations in a single cycle, without pipeline stalls.

Implementation Overview

The implementation philosophy for the LX-4080 supports the objectives of achieving excellent price/performance and time-to-market. Lexra provides two different methods of delivering the LX-4080: RTL Core for customers who use an ASIC-style development model, and SmoothCore? for customers who use a COT (Customer Owned Tooling) development model.

RTL Core: For ASIC designers, the LX-4080 RTL Core is fully scan-testable Verilog source code for the LX-4080, and can be synthesized and mapped to any ASIC vendor's standard cell or gate array library. The designer follows the ASIC vendor's design flow requirements to ensure proper sign-off. In addition to the Verilog source code. Lexra provides synthesis scripts and floor planning guidelines to maximize performance, and a system level simulation testbench.

Depending upon the design methodology, cell library, and the particular 0.35μ process

used, the LX-4080 RTL Core should achieve at least 66 MHz operation.

SmoothCore: For COT designers who have their chips manufactured at popular foundries such as Chartered, IBM, and UMC, SmoothCore is the best choice for quickest implementation, lowest cost, and best performance. The LX-4080 SmoothCore has been fully implemented and verified as a hard macro. All datapath, register file, and interfaces hve been optimized to ensure the smallest die and fastest performance possible. Lexra also provides a scan-based test pattern that achieves 99.5% fault coverage for manufacturing tests.

SmoothCore Delivery: Since the LX-4080 SmoothCore is delivered as an implemented hard macro, most of the design database will be provided by Lexra. This includes:

- Encrypted Verilog RTL and gate level simulation models
- Post-layout timing for the target process
- Behavior model for static timing analysis
- Layout GDSII
- DRC/LVS result log file

System Building Blocks

The LX-4080 is designed to fit easily into different target applications. In addition to the LX-4080 CPU core, the following integration building blocks are included:

- A Simple Memory Management Unit (SMMU)
- A Local Memory Interface (LMI) to cache, scratchpad RAM, or ROM
- Up to three Coprocessor Interfaces (CI)
- A Custom Engine Interface (CEI)
- A Lexra Bus Controller (LBC) to connect peripheral functions and secondary memories to the processor's own local buses

Simple Memory Management Unit (SMMU): The LX-4080 SMMU is designed for embedded applications that use a single address space. Its primary function is to provide memory protection between user space and kernel space. The SMMU is compatible with the MIPS address space scheme for User/Kernel modes, mapping, and cached/uncached regions. The upper bits of the physical address can be ignored to form a smaller contiguous memory space.

Local Memory Interface (LMI): The LX-4080's Harvard Architecture supports two local memory types: instruction memory and data memory. Lexra supplies a Local Memory Interface for each type. Both memories can be either asynchronous non-volatile memory (such as ROM, flash, or EPROM), or synchronous RAM. The LMI blocks are designed to interface easily with standard 32-bit wide memory blocks provided by ASIC vendors and COT library vendors.

When used as a cache memory controller, the LMI includes a two-way set associative instruction cache interface and a write-through, direct mapped data cache interface. The tag compare logic and cache replacement algorithm are provided as part of the LMI. One of the instruction cache sets can be locked down as un-swappable local memory.

Coprocessor Interface (CI): Lexra supplies an optional Coprocessor Interface for applications requiring this functionality. Up to three CIs may be implemented in one design. The Coprocessor Interface "eavesdrops" on the instruction. If a Coprocessor load (LWCz) or "move to" (MTCz, CTCz) is decoded, data will be enabled from the Data Bus into a CI register, then supplied to the designer-defined Coprocessor. Similarly, if a Coprocessor store (SWCz) or "move from" (MFCz, CFCz) is decoded, data will be fetched from the Coprocessor and loaded into a CI register, then transferred onto the Data Bus in the following cycle. The design interface includes a variable-width data bus, five-bit address and independent read and write selects for Coprocessor registers and control registers. The LX-4080 pipeline and Harvard Architecture permit single cycle Coprocessor access and transfer. Designer-defined Coprocessor condition flags (CpCondz) are synchronized by the CI then passed to the Sequencer for testing in branch instructions.

Custom Engine Interface (CEI): The LX-4080 includes a Custom Engine Interface which the designer can use to extend the MIPS ALU opcodes with application-specific or proprietary operations. Similar to the standard ALU, the CEI supplies the custom engine with two input 32-bit operands, SRC1 and SRC2. One operand is selected from the Register File. Depending on the opcode, the second operand is either selected from the Register File or is a 16-bit sign-extended immediate. The opcode is locally decoded by the custom engine and, following execution by the custom engine, the result is returned on the 32-bit RES bus to the LX-4080. To support slower or multi-cycle operations, a stall signal is supplied by the CEI. As an example, the MIPS MULT and DIV instructions are implemented as CEI functions, so designers without the need for multiply or divide do not have to pay the die size penalty.

Lexra Bus Controller (LBC): The Lexra Bus Controller is the interface between the LX-4080 and the outside world, which includes DRAM and peripherals. It is a non-multiplexed, non-pipelined, and non-parity checked bus to provide the easiest protocol for design integration. On the processor side, the LBC provides a write-buffer of configurable depth to support the write-through cache, and control for byte and half-word transfers. On the peripheral side, the LBC is designed to easily interface to industry standard bus protocols, such as PCI, USB and IEEE 1394.

The LBC can operate asynchronously at any speed from 33 MHz to the speed of the LX-4080 CPU core, or synchronously with the CPU to maximize system throughput.

Building Block Integration

To minimize the time and effort to integrate the LMI, CI, CEI, and LBC blocks with the CPU, Lexra provides the LX-4080 Configuration Tool. The Configuration Tool allows the designer to select from a menu the building blocks needed, the number of different memory blocks, target speed, target standard cell library, etc. The Configuration Tool automatically generates a reference LX-4080 top level Verilog model, makefiles, and scripts for different steps of the design flow.

For testability purposes, all building blocks include scan control signals. These signals can be gathered at the system level and multiplexed to allow the DFT software to sequentially test each building block.

Software Tool Support

The Lexra Software Developer's Kit (LSDK) allows designers to develop software-targeted designs incorporating the Lexra LX-4080. The LSDK is built around the GNU tool environment, enabling software development using the C, C++, and MIPS assembly programming languages. These tools include the GNU compiler, assembler, linker, library archive utility, and debugger. Programs developed in this environment can be loaded and run on a Verilog simulation model or on the actual LX-4080. The LSDK also includes source code for the GNU tools to allow further customization to suit designer enhancements to the LX-4080.

In addition, the LSDK also includes PMON, "The MIPS Family Prom Monitor". PMON is a public domain debug monitor designed to assist programmers of MIPS-based embedded systems to initialize the system, test, and debug their software.

The LSDK also includes a simple standard C and math library for the GNU C environment. The library was developed for embedded applications, and contains enhancements for exception handling that are optimized for the LX-4080.

Evaluation System Board

The ESB is a PCI card-based evaluation board that supports hardware prototyping and software development. The ESB includes an LX-4080 test chip, up to 32 MB of DRAM, a serial link to the Sun workstation, and a connection to the PCI motherboard. The CEI, CI, and LBC interfaces are brought out to pins on the test chip, allowing designers to prototype system level hardware.

EDA Tool Support

Lexra intends to support the most popular EDA tools, to support the widest range of design methodologies. For example, the LX-4080 supports SDF timing back annotation to simulation and timing analysis tools. Following are the EDA tools currently supported:

Design Flow	Tools Supported
Simulation	Synopsys VCS®
Synthesis	Synopsys Design Compiler®
Static Timing	Synopsys Motive®
DFT	Synopsys Sunrise®
Place & Route	Avant! Aquarius/XO

Reference Specification

The performance of the LX-4080 depends ultimately upon the silicon process and the implementation techniques and technology used. Performance also depends on the memory configuration and CI/CEI selection. The following table is provided as a reference, based upon a SmoothCore implementation of the LX-4080 CPU core only, on UMC's 0.35μ salicide process.

Specification	**LX-4080 CPU**
Transistor count	86,000
Die size (mm^2)	1.8
Frequency (worst case MHz)	100
Power dissipation (mW)	150
Operating voltage (V)	3.3
Operating environment	WC Commercial

For more information, please contact Lexra at (781) 899-5799.

LX-4080P

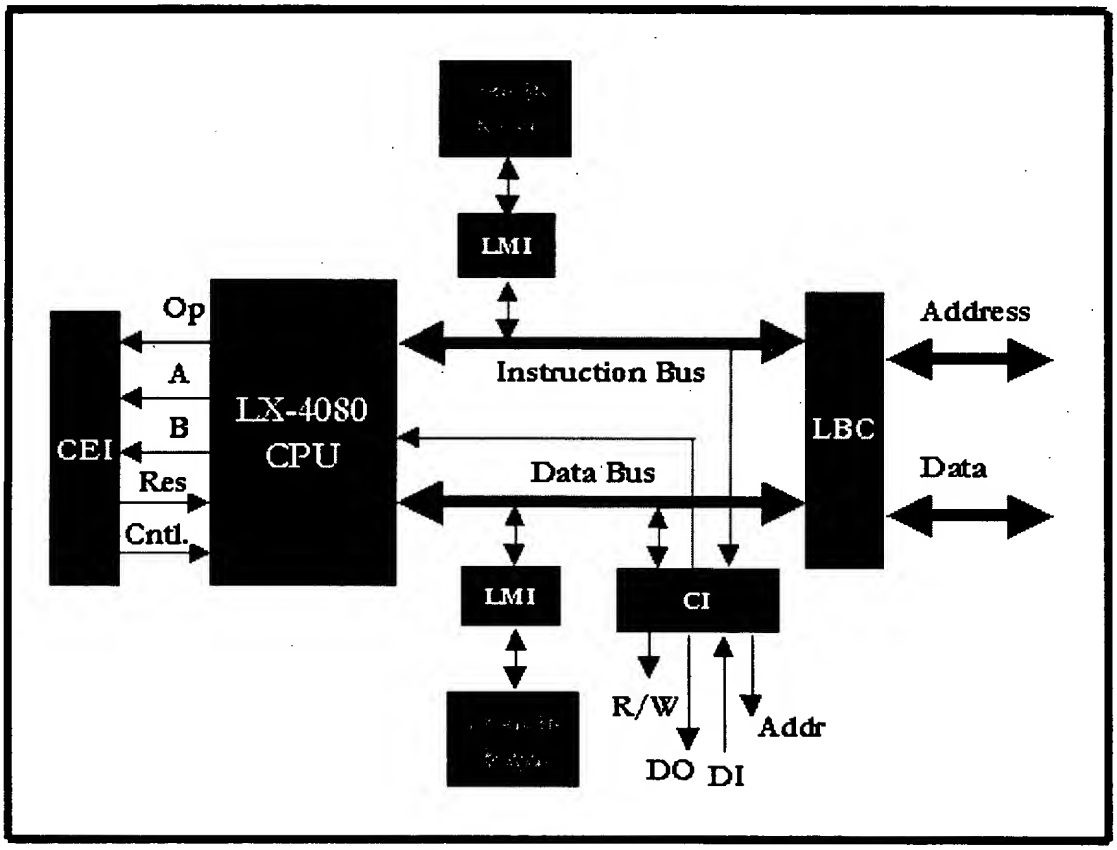
32-Bit Embedded Processor Available in a Programmable Logic Device
Product Brief

- **System Level Performance in a PLD:** Operates at 33 MHz in an Altera Flex 10K 200E device, and occupies less than 50% of available gates.
- **Supports MIPS-I® Instruction Set Architecture*:** Minimizes software porting effort and leverages broad avialability of software development tools. *Supports all MIPS-I instructions except unaligned loads and stores, which are emulated in software.
- **Easy Design:** Single phase clocking, fully synchronous design, easy to interface system bus protocol. Support for popular EDA tools.
- **Easy Migration to ASIC:** The only mainstream 32-bit processor core available in PLD and ASIC form. Allows prototyping at PCI system speeds and migration to popular ASIC vendor and COT foundries for high volume production.
- **Easy Customization:** Includes a Coprocessor Interface and the Lexra Custom Engine Interface for adding application-specific custom instructions.

trademarks of MIPS Technologies, Inc. Lexra, Inc. is not associated with MIPS Technologies, Inc. in any way.

Overview

The LX-4080P is a version of Lexra's high performance LX-4080 RISC architecture that has been optimized for the Altera Flex 10KE™ device family. The LX-4080P is the only mainstream 32-bit embedded RISC processor core that operates at 33 MHz, and can be targeted to both PLD and ASIC devices. The LX-4080P is ideal for products with critical time-to-market requirements or still-evolving standards, including data communications products, network protocol processors, cable modems, set-top boxes, printers, and next-generation disk controllers, and digital cameras.



Architecture Overview

The LX-4080P is based on Lexra's high-performance LX-4080 embedded processor architecture. The CPU core in the LX-4080P is identical to that of the LX-4080. The LX-4080P includes 1K bytes of Instruction Cache, 1K bytes of Data Cache, a four word deep write buffer, a single Coprocessor Interface, and a Custom Engine Interface.

MIPS-I Instruction Set Architecture*: The LX-4080P supports the MIPS-I programming model. The instruction set incorporates a three-port register file. Two source operands can be supplied and one destination updated per cycle. The second operand is either a register or 16-bit immediate. The instruction set includes a wide selection of ALU operations executed by the RALU, Lexra's proprietary register based ALU. The RALU also generates memory addresses for 8-bit, 16-bit and 32-bit register loads from (stores to) memory by adding a register base to an immediate offset.

Branches are based on comparisons between registers, rather than flags, and are therefore easy to relocate. Optional links following jump or branch instructions assist with subroutine programming.

*The MIPS unaligned load and store instructions are supported through software emulation only. Because these instructions are seldom used in embedded applications, a hardware implementation represents a poor price/performance tradeoff. With softare emulation, the absence of hardware support for these instructions does not affect the software programming model.

Pipeline: Instructions are executed by a five-stage pipeline, which is designed so that all transactions internal to the LX-4080 CPU core and at the interfaces occur on the positive edge of the processor clock. Two-phase clocks are not used.

Exception Handling: The LX-4080P supports the MIPS-I exception handling model. Exceptions include instruction-synchronous *traps* and hardware and software *interrupts*. All exceptions are prioritized. When an exception is taken, control is transferred to a user program located at the exception vector. The use program identifies the cause of the exception and transfers control to the application-specific handler.

Coprocessor Operations: The LX-4080P supports all 32-bit Coprocessor operations. These include moves to and from the Coprocessor general registers and control registers (MTCz, MFCz, CTCz, CFCz), Coprocessor loads and stores (LWCz, SWCz) and branches based on Coprocessor condition flags (BCzT, BCzF). The LX-4080P includes a Coprocessor Interface which can support all Coprocessor Operations in a single cycle, without pipeline stalls

System Building Blocks

In addition to the LX-4080 CPU core, the LX-4080P includes the following blocks:

- **A Simple Memory Management Unit (SMMU):** The SMMU is designed for embedded applications that use a single address space. Its primary function is to provide memory protection between user space and kernel space. The SMMU is compatible with the MIPS address space scheme for User/Kernel modes, mapping, and cached/uncached regions. The upper bits of the physical address can be ignored to form a smaller contiguous memory space.

- **Two Local Memory Interfaces (LMI):** One that interfaces to 1K bytes of Instruction Memory and one that interfaces to 1K bytes of Data Memory

- **One Coprocessor Interface (CI):** The Coprocessor Interface "eavesdrops" on the instruction. If a Coprocessor load (LWCz) or "move to" (MTCz, CTCz) is decoded, data will be enabled from the Data Bus into a CI register, then supplied to the designer-defined Coprocessor. Similarly, if a Coprocessor store (SWCz) or "move from" (MFCz, CFCz) is decoded, data will be fetched from the Coprocessor and loaded into a CI register, then transferred onto the Data Bus in the following cycle. The design interface includes a variable-width data bus, five-bit address and independent read and write selects for Coprocessor registers and control registers. The LX-4080 pipeline and Harvard Architecture permit single cycle Coprocessor access and transfer. Designer-defined Coprocessor condition flags (CpCondz) are synchronized by the CI then passed to the Sequencer for testing in branch instructions.

- **A Custom Engine Interface (CEI):** The designer can use the CEI to extend the MIPS ALU opcodes with application-specific or proprietary operations. Similar to the standard ALU, the CEI supplies the custom engine with two input 32-bit operands, SRC1 and SRC2. One operand is selected from the Register File. Depending on the opcode, the second operand is

either selected from the Register File or is a 16-bit sign-extended immediate. The opcode is locally decoded by the custom engine and, following execution by the custom engine, the result is returned on the 32-bit RES bus to the LX-4080. To support slower or multi-cycle operations, a stall signal is supplied by the CEI. As an example, the MIPS MULT and DIV instructions are implemented as CEI functions, so designers without the need for multiply or divide do not have to pay the die size penalty.

- **A Lexra Bus Controller (LBC):** The LBC connects peripheral functions and secondary memories to the processor's own local buses. It is a non-multiplexed, non-pipelined, and non-parity checked bus to provide the easiest protocol for design integration. On the processor side, the LBC provides four word deep write buffer, and control for byte and half-word transfers. On the peripheral side, the LBC is designed to easily interface to industry standard bus protocols, such as PCI, USB and IEEE 1394.

Implementation Overview

The LX-4080P includes:

- A Max+Plus II database optimized for the Altera Flex 10KE device family
- Encrypted Verilog RTL and gate level simulation models
- Post-layout timing

Software Tool Support

The Lexra Software Developer's Kit (LSDK) allows designers to develop software-targeted designs incorporating the Lexra LX-4080P. The LSDK is built around the GNU tool environment, enabling software development using the C, C++, and MIPS assembly programming languages. These tools include the GNU compiler, assembler, linker, library archive utility, and debugger. Programs developed in this environment can be loaded and run on a Verilog simulation model or on the actual LX-4080P. The LSDK also includes source code for the GNU tools to allow further customization to suit designer enhancements to the LX-4080P.

In addition, the LSDK also includes PMON, "The MIPS Family Prom Monitor". PMON is a public domain debug monitor designed to assist programmers of MIPS-based embedded systems to initialize the system, test, and debug their software.

The LSDK also includes a simple standard C and math library for the GNU C environment. The library was developed for embedded applications, and contains enhancements for exception handling that are optimized for the Lexra architecture.

Evaluation System Board

The LX-4080P Evaluation System Board is a PCI card-based evaluation board that supports hardware prototyping and software development. The ESB includes the LX-4080P implemented on an Altera Flex 10K 200E PLD, up to 32 MB of DRAM, a serial link to the Sun workstation, and a connection to the PCI motherboard. The CEI, CI, and LBC interfaces are brought out to pins on the PLD, allowing designers to prototype system level hardware.

EDA Tool Support

Lexra intends to support the most popular EDA tools, to support the widest range of design methodologies. Following are the EDA tools currently supported for the LX-4080P:

Design Flow	Tools Supported
Simulation	Synopsys VCS®
Synthesis	The LX-4080P has been pre-synthesized using Synplicity Synplify®
Gate Level Simulation, Timing, and Place & Route	Altera Max+Plus II®

For more information, please contact Lexra at (781) 899-5799.

Argonaut RISC Cores

ARC Overview

ARC is a powerful and flexible RISC microprocessor solution for embedded applications. Provided as a ``soft macro" which is fully supported with industry standard tools, ARC provides customers with a cost-effective design ready-to-use or which you can configure to meet your performance and cost needs.

Last year, nine semiconductor companies licensed the ARC microprocessor and ARC was selected for twenty-three new designs, a strong start for a newcomer to the industry. Specifically, customers are choosing ARC as a way of:

- Reducing costs
- Improving time to market
- Improving performance (including reducing power consumption)
- Adding features to their designs
- Getting access to the best development tools available
- Being process and foundry independent

The fastest way to design..

..use an ARC

The difference is the architecture

For instance, some customers who were previously using a microprocessor core augmented with a DSP are moving to an ARC with a few customised instructions saving themselves nearly $10 of manufacturing costs per unit. Other significant savings can be achieved by the capability to move to newer manufacturing technologies such .25 or .18 micron with a simple re-synthesis rather than the lengthy (perhaps six months) process required when working with hard macros.

ARC offers a complete hardware and software development tool chain which features the ARC Install Wizard, a graphical tool which configures an ARC with the custom instructions, cache, interfaces, and your chosen target technology.

As ASIC designers ourselves, we are very aware that to be broadly useful, a processor requires a complete and integrated development system, documentation, software, and testing support. ARC offers a hardware development system (Altera-based), an industry standard software toolchain (from Metaware), an emulation environment, and an RTOS (from ThreadX). The combination of the ARC's flexibility, Altera's leading edge FPGA technology, ThreadX's RTOS, and MetaWare's powerful software development environment provide a unique low-cost way of rapidly developing complete systems. These tools provide the unique capability of evaluating various processor configurations impact on software performance so that the entire system's performance can be optimised. The ARC's development tools also facilitate simultaneous software and hardware development.

We at Argonaut RISC Cores are proud of how we are helping customers achieve stunning design breakthroughs. ARC is providing competitive advantage for designs in the wireless, communications, disc drive, 3D graphics, and multimedia markets.

Thank you for taking the time to learn more about the ARC Microprocessor. By helping our customers differentiate their design through the use of a flexible ARChitecture, we aim to remain the leader in reusable semiconductor IP for the embedded RISC market.

ARC Benefits

- Very High Performance
- Scalable across Processes
- Small die size
- Reduced Memory Requirements
- Controllable Power consumption
- Process Independent
- Easy to integrate into your ASIC

ARC offers Very High Performance versus other RISC or CISC processors without having to invest multiple man years in hand crafting. In a good .35 CMOS process ARC will achieve 100 MHz.

ARC is Scalable across Processes which means you can migrate a design from one process to the next and see performance improvement of 50% to 100%. These performance improvements can be achieve by using the synthesis scripts that are provided with the technology.

ARC offers Small Die Sizes that are typically 50% smaller than most RISC

processors. The average size of ARC in a telecommunications, multimedia or wireless application is 25k gates.

ARC requires much less memory than a fixed instruction general purpose processor for two reasons. First ARC is a single cycle processor and performs most instructions in one clock cycle. Second, because ARC is Application Specific it does not require a lot of memory for buffering like a fixed instruction processor. With ARC you can typically reduce the amount of memory required versus any other RISC or CISC processor by 50% to 75%.

In applications like wireless where power consumption is a critical factor, ARC can be fine tuned through the synthesis scripts to achieve the power consumption that is best for your product. ARC has been implemented in various 3v and 2v processes.

ARC is a fully synchronous design and therefore can be easily migrated to a new process without the need to invest costly development time.

ARC has been architected to be a master or slave processor. Integration into your design is easy to learn and simple to use. I cache and D cache are user definable, there is a host interface to integrate into your PCI, S or proprietary BUS. The designer can use Scratchpad Memory inside ARC for storing results from special functions like a MAC and there are ``Magic Registers" that can be use as pointers.

ARC is ARChitected to help you differentiate your product from your competition and to help you satisfy your customers' requirements. ARC is a total solution that is designed to ``Help you Succeed".

Key Features

1. Technology independent 32-bit RISC

2. Small core size

- <2 sq. mm or 16k gates.
- 3LM 0.35um CMOS, using RAM cell for register block.

3. High clock speed - 100MHz+ (0.35um CMOS Standard Cell)

4. Defined in VHDL for synthesis

5. Flexible design allows easy customization for your application

- synthesize for minimum area or maximum speed.
- select instruction cache size
- select the instructions required for your application
- select register file size
- add special-purpose registers
- add new condition code choices
- include a local scratchpad RAM

- Library of multimedia extensions for 3D & DSP applications

6. Development tools

- MetaWare High C Toolchain, Express Logic ThreadX RTOS
- Designed for low-cost, high performance embedded applications
- Host port for accessing ARC registers from another CPU
- External single step and halt lines

ARC Technical Details

- Pipelined 32-bit RISC Microprocessor
- 32 x 32 bit general purpose registers as standard
- 4 stage pipeline
- 32-bit address space for data
- 24-bit address space for code
- Basic core contains arithmetic and logic operations
- 2 and 3-operand instructions
- 9-bit sign-extended or 32-bit immediate data
- Delayed branches
- Conditional execution of regular instructions
- Optional flag setting
- Zero overhead loop feature
- Scoreboarded delayed loads from memory
- Loads and Stores support 32/16/8 bit data at any 32-bit address
- Address writeback for table traversal in memory
- Access local RAM and control registers via special load & store instructions.

Extension possibilities

- 16 slots available for additional 3-operand instructions
- 55 slots available for additional 2-operand instructions
- 32 free positions in the general register space
- allows up to 60 general-purpose 32-bit registers
- add special-purpose registers for zero latency access

Auxiliary register space provides a 32-bit address space separate from the memory system

- Add local scratchpad RAM
- Add special registers
- Add up to 16 extra condition code choices

Extension Library includes:

- 32 x 32bit scoreboarded multiply block
- Single cycle 32bit barrel-shifter/rotate block
- Normalize (find-first-bit) instruction
- Mirror the core's arithmetic and logical instructions, but send results directly to a command buffer, not the register file.
- Local scratchpad RAM, with block move to memory.
- 16-bit MUL/MAC block, 36-bit accumulator.

- Sliding pointer access to local SRAM, using linear arithmetic.

ARC Design Methodology

ARC Design Methodology

The design flow for using ARC is very similar to the standard ASIC design flow. The diagrams on the following pages show the flow for both the hardware and software development.

The development team decides what functions will be implemented in hardware and what functions will be implemented in software.

Custom instructions are defined, specified and developed.

The Base RISC Engine, Functions and Custom instructions are synthesized using the Synopsys Altera Library.

The gate level version of the design is then implemented into the Hardware Prototyping System by by programming the Altera FLEX 10k100 PLD. I-cache can also be supported by the Prototyping System, the size can be between 512 bytes and 16 K bytes of cache.

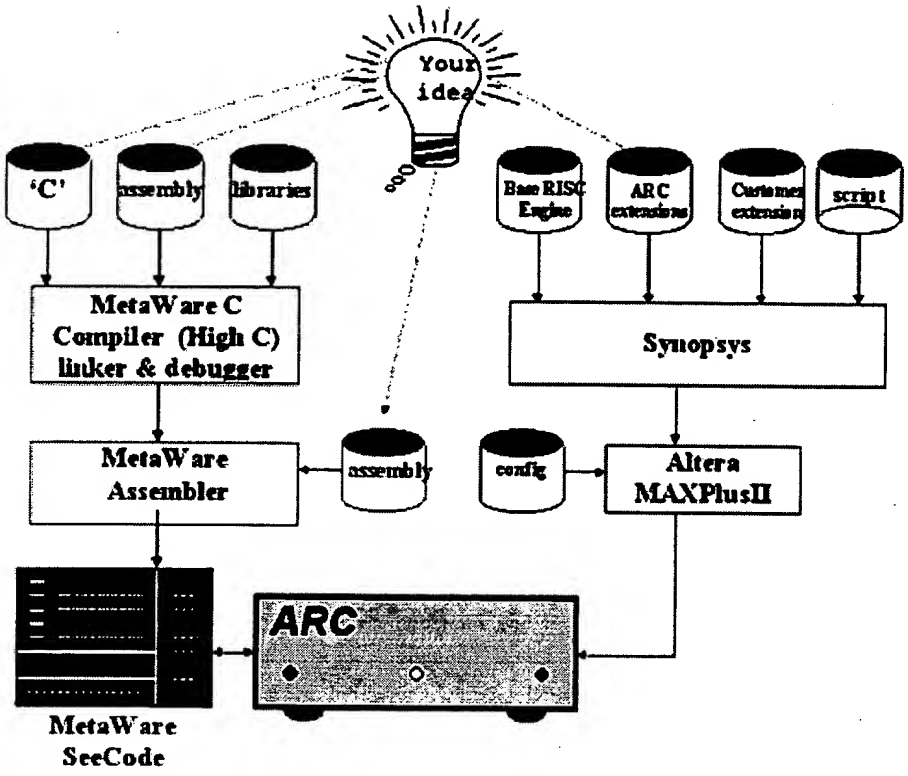
Simultaneously, the software developers can compile code using MetaWare's `C' compiler and assembler. Linkers, debuggers and optimizers are all available from MetaWare to complete the software development flow.

With the VHDL design, the Hardware Prototyping System can be programmed in three to four hours thereby allowing the team to easily and quickly test, make changes and observe the results.

Once the Hardware Prototyping System has been programmed the software developers can plug the system into a PC and begin running actual code through the system to analyze and observe the results. If the expected results are not achieved, changes can be made quickly and at no cost thereby encouraging experimentation.

ARC's ASRP Architecture is easy to use and the design flow is consistent with a top-down ASIC methodology. As a result it takes about 6-10 hours to train an experienced ASIC design team how to use the ASRP Architecture and ARC.

ARC Design Methodology Diagram

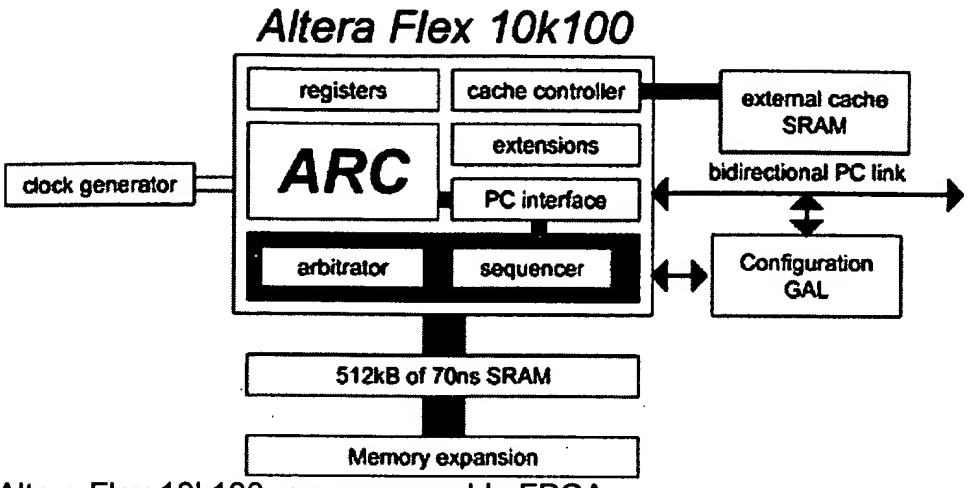


- Complete ARC System Proof-of-Concept
- Develop Custom Code at Reasonable Clock Speeds
- Prototype Designs of ARC Extensions
- Powerful Hardware/Software Code design Environment
- Low Cost Development Environment

Hardware Prototyping Development System

ARC's Hardware Prototyping Development System is unique because it can be reprogrammed in three to four hours, provides the ability to test code in a hardware environment at 3.125 MHz and is highly flexible. Altera's FLEX 10k100 can handle 40k to 50k gate instantiation while the typical size of ARC in a Multimedia, Telecommunications or wireless application is around 25k gates. Also, Altera has announced plans to offer the next generation of the Flex part in a .35 micron process that should double the performance.

The Hardware Prototyping Development System is available from Argonaut or your ASIC vendor for $5,000. MetaWare's entire High C software development environment is available on PC or UNIX and costs between $4,995 and $9,995. This means a complete hardware and software development environment for ARC can be less than $10,000.



- Altera Flex 10k100 reprogrammable FPGA
- System Clock 3.125MHz
- 512Kb of 70ns SRAM on-board
- Expansion connector on memory system bus, up to 16Mb memory space
- 32Kb of 15ns SRAM, used as cache memory space
- External interupt inputs
- Altera configuration from PC parallel port
- Buffering for bi-directional PC parallel link
- Reset Button
- `Power', `ARC running' and `PC link active' LEDs

'C' Development Tools

'C' Development Tools

Reliable, high quality tools are essential for any development system. Argonaut considered many suppliers before contracting the development of the ARC 'C' development environment.

The criteria for selection were reliability, completeness, useability, flexibility and cost effective user support.

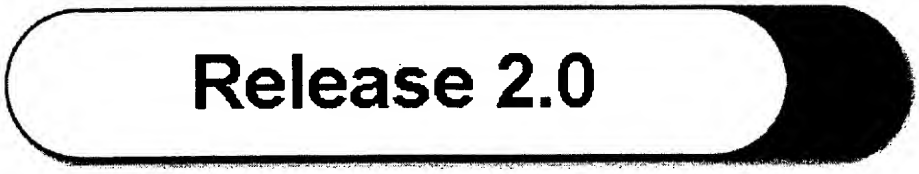
We chose MetaWare because of their expertise in the embedded market and their outstanding service and support. They provide a high quality complete solution (compiler, linker, debugger, profiler) and have customised their tools to support the advanced architecture of the ARC. Their technical support program is tailored to fulfill any project size, they can support a large project for an ASIC vendor or a small project for an end user.

Metaware's website can be found at www.metaware.com

Future Developments

The next release of the Metaware compiler (Q3-1998) will include:
•C++ / embedded C++ •RTOS support within the debugger •improved compiled code size •improved compiled code speed

Argonaut and Metaware are working together to continually improve the quality and flexibilty of the tools.



ARC Release 2.0

Key Features

- **NEW** GUI and Command-Line ARC builder
- **NEW** Extensive and improved overall product test system
- **NEW** Library of Extension Instructions
- **NEW** Library of memory subsystems
- **NEW** Updated product documentation

GUI and Command-Line ARC builder that can build:

- Core ARC - Excludes memory subsystems, host i/f and comms channel
- ARCAngel - Builds ARC System to be used on ARCAngel Development System
- Generic ARC - Includes memory subsystems, host i/f and comms channel

Available Memory Subsystems

- Direct Mapped I-Cache
- **NEW** Template for instruction code ROM/RAM (Used instead of I-Cache)
- Synthesised or 3-port RAM for register file
- **NEW** Stack Accelerator based on fast RAM accessed through load/store interface
- **NEW** Local Scratchpad RAM, accessed through auxiliary interface
- **NEW** Block move between Local Scratchpad RAM and main memory
- **NEW** Sliding pointer access to Local Scratchpad RAM using linear arithmetic
- **NEW** Memory Arbitration Block
- **NEW** SRAM memory sequencer

Available Extension Instructions

- **NEW** Fast 32 x 32 bit scoreboarded multiply instruction
- Small 32 x 32 bit scoreboarded multiply instruction
- Single cycle 32 bit barrel-shifter/rotate instruction
- Min/Max instruction
- Normalise (find-first-bit) instruction
- Swap instruction
- 16-bit MUL/MAC instruction, 36-bit accumulator

Host i/f and Comms Channel

- **NEW** PC/Sun Parallel Port Comms Module
- **NEW** Template for custom Comms Module

Other Features

- Customisable reset and interrupt vectors
- Main memory model reads HEX files generated from Metaware compiler
- RAM models with timing information

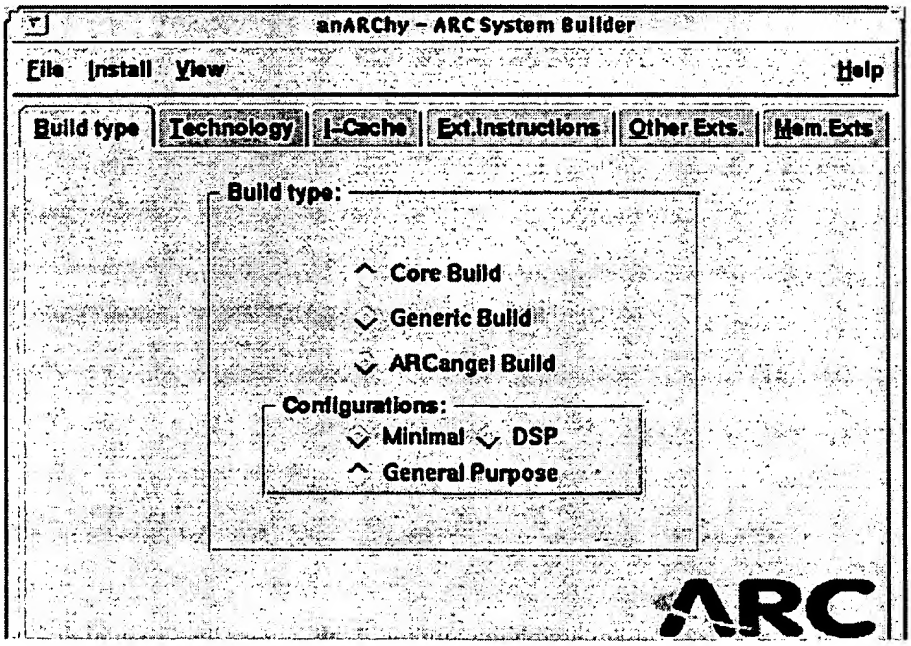
ARC Configuration Wizard

ARC System Builder ``Configuration" Wizard

The ARC Wizard has been developed to enable fast and efficient system designs built around the ARC microprocessor architecture, allowing an engineer to build their own microprocessor(s) as a simple point and click exercise.

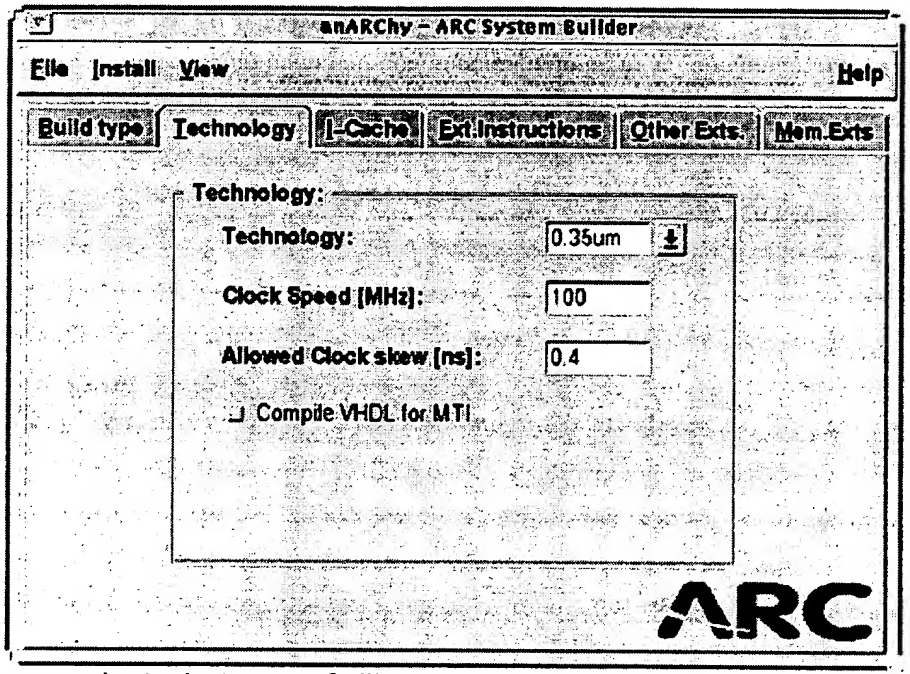
This powerful tool frees up the engineer from the normal tedious tasks involved in using microprocessor cores, allowing the skilled engineer to exploit the ARC to its full potential, by producing a much better product in less time which is smaller, faster and more flexible than they could achieve with traditional systems.

The following screen shots show the various options that are available with Release 2.0 of the ARC Systems builder.

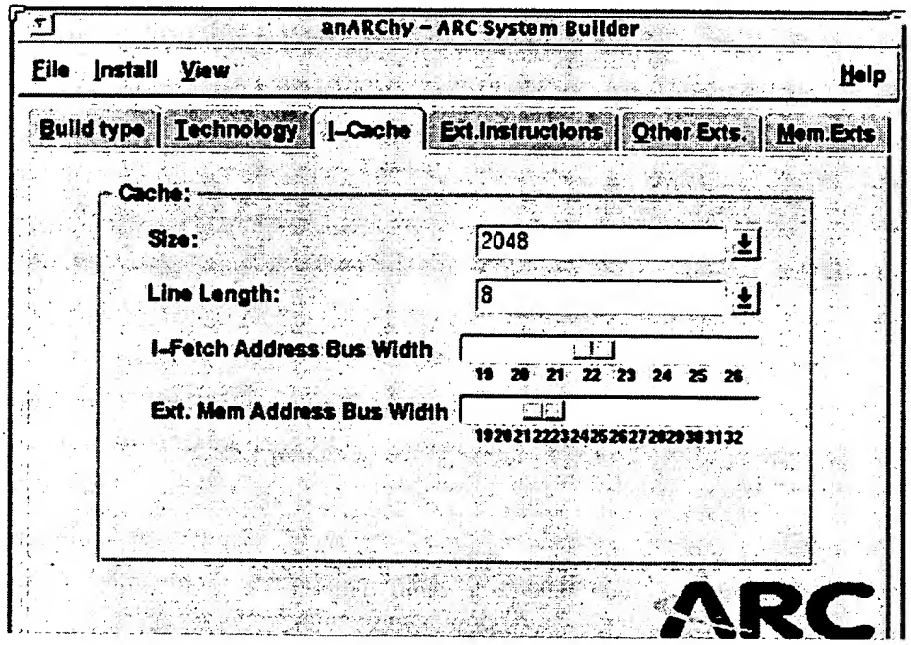


Here we can select what type of build, eg for our development board, a complete system or a basic core.

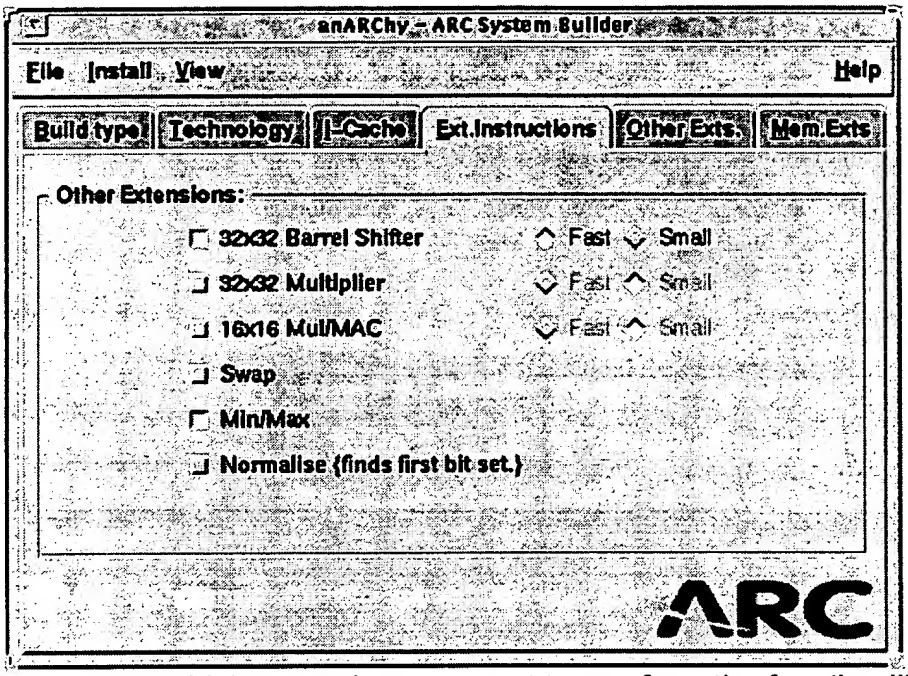
We can then select from an expanding list of pre-defined configurations. These configurations add the functionality to the basic core, so in the case of the DSP option we add DSP functions to the ARC. This now gives us a RISC microprocessor that has the performance characteristics of a DSP, the programmability characteristics of a RISC at a smaller size than other 32bit RISC cores, removing the need to have a DSP and a separate microprocessor and all the associated memories for code and data for two cores. The silicon area and associated power saving benefits are obvious here.



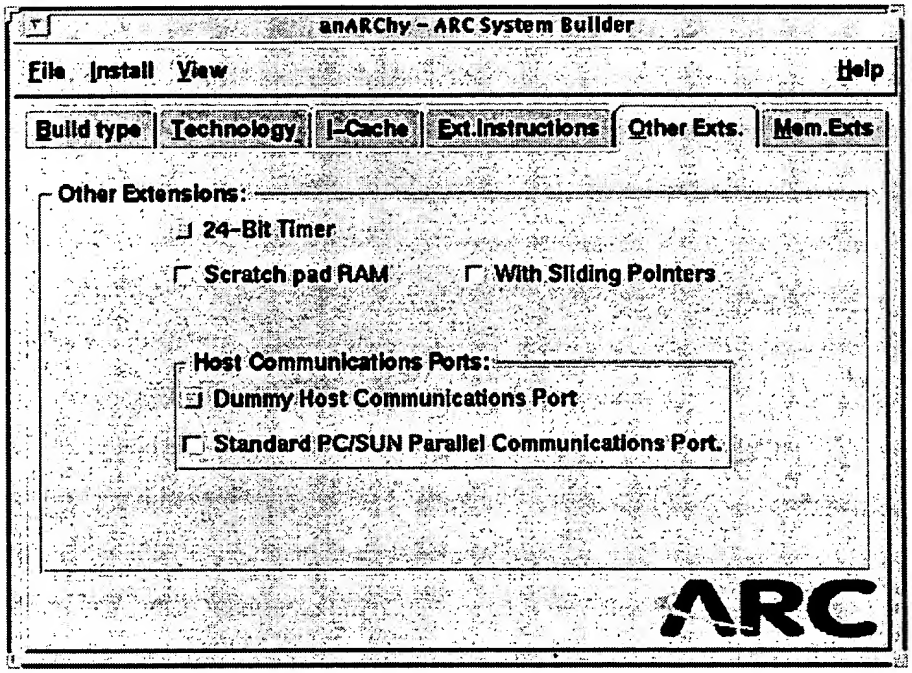
Here we can select what type of silicon technology we want to use, and the desired clock speed. These are used for setting up the synthesis scripts.



Next we set up the Instruction Cache size we want or none at all. Using the ARCangel prototyping system we can try out different configurations so that we always use the smallest amount possible for your application.

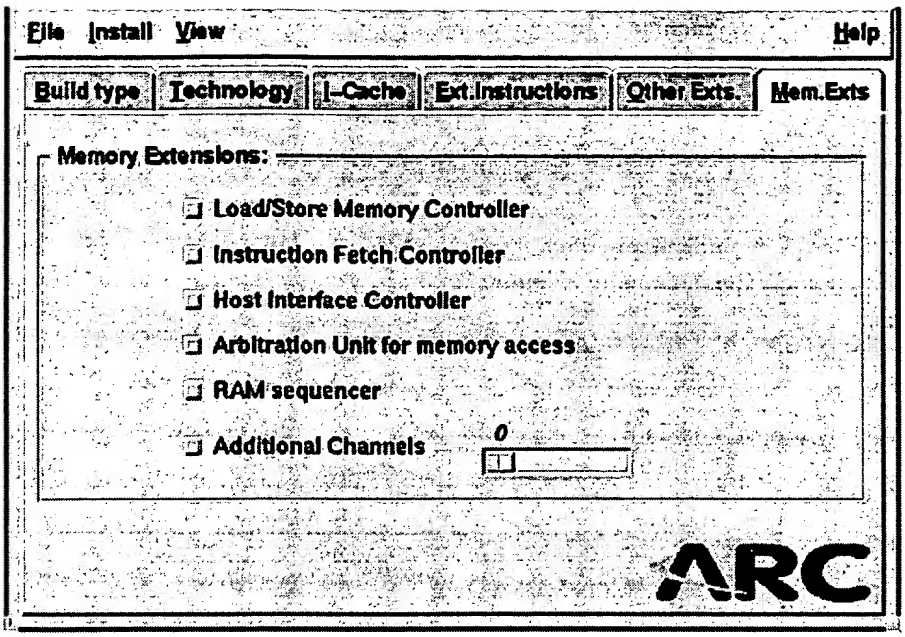


Here we can select which extensions we want to use from the function library. Defaults will be set from the configurations page.



And finally we select which memory channels and how many we need in our design.

All that's left to do is select install, and all the files will be set up on your system, complete with file hierarchy and Synthesis scripts.


File Install View
Help
Build type
Technology
I-Cache
Ext.Instructions
Other Exts.
Mem.Exts
Memory Extensions:
Load/Store Memory Controller
Instruction Fetch Controller
Host Interface Controller
Arbitration Unit for memory access
RAM sequencer
Additional Channels
0
ARC